MODELO DE PREDICCIÓN DE FALLOS PARA PROYECTOS DE SOFTWARE DE LA UNIVERSIDAD TECNOLÓGICA DE PEREIRA UTILIZANDO REDES NEURONALES

Ing. Bibiana Patricia Arias Villada

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
Facultad de Ingeniería industrial
Maestría en Investigación de Operaciones y Estadística
Pereira
2015

MODELO DE PREDICCIÓN DE FALLOS PARA PROYECTOS DE SOFTWARE DE LA UNIVERSIDAD TECNOLÓGICA DE PEREIRA UTILIZANDO REDES NEURONALES

Ing. Bibiana Patricia Arias Villada

Trabajo de investigación presentado como requisito para optar al título de:
Magister en Investigación de Operaciones

Director
M.Sc. Wilson Arenas Valencia

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
Facultad de Ingeniería industrial
Maestría en Investigación de Operaciones y Estadística
Pereira
2015

Nota de aceptación:

_____

_____

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Firma del jurado

_____

Firma del Jurado

Resumen

Uno de los objetivos más importantes en el desarrollo de software es la estimación de posibles fallos de un proyecto antes de ser entregado al usuario. A pesar de que previo a la etapa final se hacen pruebas de carga, rendimiento y seguridad al software, se pueden presentar fallos cuando el módulo entra en funcionamiento.
En este trabajo se propone un modelo de predicción de defectos para proyectos de software utilizando la técnica de Redes Neuronales y recogiendo los datos necesarios para llevar a cabo las predicciones. Los datos seleccionados dan una amplia idea sobre los errores de varios productos de software ya que sus valores contienen información importante respecto a su ciclo de vida.
El concepto de sistemas de predicción de defectos en proyectos de software antes de entrar en funcionamiento será cubierto a través de esta investigación.

Este trabajo desarrolla una herramienta basada en redes neuronales y entrenada con métricas de proyectos de software hechos por la División de sistemas de la Universidad Tecnológica de Pereira con el fin de predecir la tendencia a fallos de cada nuevo módulo, y convertirla en una herramienta útil para la toma de decisiones por parte de la alta dirección acerca de la calidad de cada desarrollo de software.

Palabras clave: *Red Neuronal Artificial, Módulo de software, Perceptron Multicapa, Predicción, Propagación hacia atrás, Variables de Medición.*

Abstract

One of the most important goals in software development is the estimate of possible failure of a project before it becomes operational, although prior to this stage load testing, performance and security are made, failures may occur when the module enters production.

This paper presents a defect prediction model for software projects using the technique of Neural Networks and collecting the data necessary to carry out the predictions is proposed.

The selected data give a comprehensive picture about the errors of various software products because their values contain important information on its life cycle.

The concept of prediction systems defects in software projects and the difference between these will be covered by this investigation.

This paper develops a software tool based on neural network and trains it using data of software projects developed by Systems Division of Pereira Technological University to predict the tendency to errors of each new module, , in order to make it a useful tool for decision- making by senior management about the quality of each software development.

Keywords: *Artificial Neural Network, software module, Multilayer Perceptron, Prediction, backpropagation, Variable Measurement.*

Contenido

Lista de Figuras

Lista de Ecuaciones

Lista de Tablas

Lista de Anexos

MODELO DE PREDICCIÓN DE FALLOS PARA PROYECTOS DE SOFTWARE DE LA UNIVERSIDAD TECNOLÓGICA DE PEREIRA UTILIZANDO REDES NEURONALES

1. GENERALIDADES

En este capítulo se mostrarán los elementos y necesidades que motivaron esta investigación, delimitación y alcance del proyecto y los recursos utilizados.

1.1 Introducción:

Aunque existen varios métodos formales y herramientas para automatizar un software, la mayor parte de su desarrollo depende de decisiones humanas. Por lo tanto, los errores conocidos como “bugs” son en su mayoría introducidos como consecuencia del factor humano.

La calidad del software está directamente relacionada con su número de defectos, minimizar este número requiere de una “prueba” completa y precisa. Los modelos de predicción de defectos son herramientas útiles para realizar pruebas a un software, estimaciones precisas de los módulos[1] defectuosos pueden producir disminución de tiempos de prueba, y pueden beneficiar a los administradores de proyectos en términos de asignación de recursos de una forma más eficaz.

Los investigadores han centrado sus investigaciones en modelos de predicción de propensión a fallos (PPF), estos modelos tienden a predecir que partes del software desarrollado recientemente pueden ser más propensos a fallas. De acuerdo a los modelos de PPF, un sistema consta de los componentes descritos por métricas (propiedades numéricas), tales como la complejidad del código o el número de cambios pre-lanzamiento. Algunos modelos estadísticos, como árboles de decisión o regresión logística, utilizan estos parámetros para predecir la tendencia a errores. Inicialmente, se entrena el modelo estadístico con los datos

[1] En programación, los módulos son partes de un software que hacen una tarea específica y que pueden ser compilados por separado, esto los hace reusables. Gracias a la programación por módulos múltiples programadores de un mismo proyecto pueden trabajar en diferentes módulos en forma simultánea, produciendo ahorro en los tiempos de desarrollo.

de un software similar para el cual conocemos las métricas de los componentes. Una vez entrenada la herramienta la podemos utilizar para predecir la propensión del nuevo sistema a los fallos.

1.2 Motivación

En la industria del desarrollo de software, todas los casos de desarrollo se encuentran en una fuerte competencia, si se minimiza el tiempo de desarrollo se disminuye el costo total del proyecto. Por otro lado, menos tiempo de desarrollo y pruebas también aumenta la proporción de densidad de defectos en términos del producto final. Por lo tanto, una solución de predicción defecto puede proporcionar las métricas necesarias para tomar una decisión relacionada con la entrega del producto. La alta dirección puede tomar una decisión sobre el lanzamiento de un producto si el nivel de densidad de defectos es inferior a un determinado umbral.

1.3 Planteamiento del Problema

1.3.1 Descripción del Problema

En la actualidad el software no es solo una herramienta básica de trabajo sino que ahora está presente en los quehaceres de la vida cotidiana, dotados cada vez de más funcionalidades lo que hace su desarrollo mucho más complejo. Esto conlleva a un aumento de errores en su utilización debido a estimaciones incorrectas durante su elaboración, lo que pueden acarrear consecuencias en términos económicos y de tiempo. Son muchos los esfuerzos para predecir el número de defectos de un sistema y la estimación de su fiabilidad, las métricas de software son los atributos de software que ayudan a entender las diferentes características que lo componen. Dichas métricas se utilizan sobre todo con el objetivo de garantizar la calidad del producto, la eficiencia del proceso y sus riesgos. Uno de los beneficios más importantes de las métricas del software es que proporcionan información para la predicción de fallas. En la actualidad, existen

numerosos indicadores que pueden ser utilizados por los coordinadores del proyecto para evaluar los riesgos del software. Las primeras investigaciones sobre las métricas del software han centrado su atención principalmente en número de líneas de código (LOC), estas investigaciones intentan detectar defectos utilizando métricas relacionadas con la extensión del código fuente o su complejidad lo que las hace no muy exactas ya que el número de errores descubiertos está relacionado con la cantidad de pruebas realizadas.

El número de defectos deben ser interpretados en el contexto de los esfuerzos de la prueba, de lo contrario, el número de defectos llevaría a la conclusión equivocada. Por ejemplo, si tenemos un programa complejo que no se prueba, el número de defectos de ese programa será cero. En este caso, el número de defectos no dicen nada acerca de la calidad del software, ya que la prueba no se realizó.

1.3.2 Elementos del Problema

- Aplicaciones cada vez más complejas.
- Carencia de datos históricos de proyectos ya terminados.
- Poco análisis de los datos existentes.
- Técnicas de estimación inexactas.
- Diferencias entre los modelos resultantes en cada etapa de desarrollo.
- Documentación incompleta e inexacta lo que genera software poco flexible.
- Pruebas incompletas
- Herramientas de decisión imprecisas.

1.3.3 Delimitación del Problema

El conjunto de datos a analizar son obtenidos de los desarrollo hechos por la División de Sistemas de la Universidad Tecnológica de Pereira en los últimos 7 años.
La técnica a utilizar es Redes Neuronales, se desarrollará una herramienta en un lenguaje de programación Delphi y se probará con datos de otros proyectos realizados los cuales están documentados en artículos y se compararan los resultados, luego con la herramienta ya entrenada se utilizaran los datos aportados por la División de Sistemas y se analizarán los resultados.

1.3.4 Formulación del problema

¿Puede un modelo de predicción basado en Redes Neuronales ser una herramienta eficaz de estimación de fallos de los diferentes proyectos de software de la Universidad Tecnológica de Pereira?

1.4 Justificación

Los métodos de estimación del software han evolucionado en los últimos años, pasando de procedimientos pocos formales como la opinión de expertos hasta completos modelos matemáticos de estimación que relacionan múltiples variables. El salto más significativo ha sido el paso de los modelos de regresión a modelos inducidos con algoritmos de aprendizaje automático. La superioridad de estas técnicas se debe fundamentalmente a la generación automática de un modelo de predicción en el que aparecen los atributos más influyentes en la estimación de la variable objetivo sin necesidad de formular una hipótesis. La mayoría de los método clásicos de estimación usan un modelo paramétrico en forma de expresión matemática que calibran a partir de datos históricos, en cambio, los algoritmos de aprendizaje automáticos proporcionan modelos no paramétricos sin que sea necesario realizar presunción alguna sobre la forma de función que relaciona las variables implicadas en la estimación. En estos métodos, la hipótesis es el propio

modelo inducido automáticamente a partir de los datos en el algoritmo de aprendizaje.

Por otra parte, los enfoques algorítmicos o paramétricos son generalmente inapropiados para modelar las complejas relaciones entre las variables que existen en muchos entornos de desarrollo de software, por lo que producen resultados inexactos incluso habiendo realizado la calibración del modelo (Cosín 2007). La minería de datos se comprende de una serie de técnicas, algoritmos y métodos que imitan la cualidad humana del aprendizaje: ser capaz de extraer nuevos conocimientos a partir de las experiencias (entrenamiento).

El fin de la minería de datos es la explotación de grandes volúmenes de datos con vistas al descubrimiento de información previamente desconocida y que pueda servir de ayuda en el proceso de toma de decisiones, formando parte del conjunto de tecnologías de la Inteligencia de Negocio (realización eficiente de todas las actividades relacionadas con la generación, extracción, organización, análisis, compartición y distribución del conocimiento de una organización) (Villena s.f.).

1.5 OBJETIVO GENERAL

Generar un modelo utilizando la técnica de Redes Neuronales que permita estimar la tendencia a fallos de proyectos de software desarrollados en División de Sistemas de la Universidad Tecnológica de Pereira.

1.6 OBJETIVOS ESPECÍFICOS

1. Analizar los trabajos y modelos ya existentes acerca de predicción de software.
2. Utilizar la técnica de redes neuronales para crear una herramienta de evaluación del software.
3. Entrenar el aplicativo con los datos históricos de desarrollos anteriores de software de la División de Sistemas de la Universidad Tecnológica de Pereira para luego predecir la tendencia a errores de un desarrollo nuevo.

4. Analizar los resultados para validar la efectividad de la herramienta.
5. Proponer un nuevo modelo de evaluación de software.

1.7 Metodología

1.7.1 Parámetros, Procedimientos y Técnicas de la investigación

Se desarrolló una herramienta de software basada en redes neuronales y entrenarda con datos de proyectos de software hechos por la División de sistemas de la Universidad Tecnológica de Pereira, para luego contrastar estos datos y verificar la efectividad de la herramienta, esto con el fin de que se convierta en una herramienta útil para la toma de decisiones por parte de la alta dirección acerca de la calidad de cada software.

1.8 Análisis de la Información

El objetivo principal de la investigación es predecir defectos en el desarrollo de módulos de software. Por lo tanto, la recopilación de datos y una definición clara de los “bugs” son dos de los factores críticos para el éxito de una predicción. Las listas de control de errores son las principales fuentes para el análisis, algunos sistemas de control de defectos pueden llevar a un riesgo ya que alguien fácilmente puede ingresar un tipo de error sin ningún tipo de aprobación o de filtro. Por lo tanto, ningún sistema público puede ser considerado como un sistema fiable de seguimiento en el ámbito de esta investigación.

Al seleccionar el sistema de seguimiento de fallos, también es importante que se incluyan las siguientes mediciones: tipo de error, módulo de software en el que se observa el error, la clasificación de los errores (interfaz de usuario, la función interfaz, y otros), y la gravedad del error. Un error puede provocar un fallo total de un sistema, mientras que otro sólo puede degradar el rendimiento. En contexto, la gravedad del fallo describe el impacto de un error en el sistema de software.

El sistema de predicción propuesto será capaz de predecir no sólo a nivel global, sino también a nivel de módulo.

Asumimos que cada módulo estable puede contener errores, y todos han sido detectados. Este supuesto es válido, ya que el peso de los errores se da según su gravedad, y podemos pasar por alto los que tienen bajo impacto. En consecuencia, el error fijación de la información es el dato fundamental para predecir la estabilidad de próximas versiones.

Una propiedad importante de los datos “bugs” es el nivel de gravedad. Los errores son analizados con sus niveles de gravedad para llegar a la densidad de defectos de los módulos. Cada módulo de un proyecto cuenta con grandes diferencias a los demás en términos de corrección de errores y función.

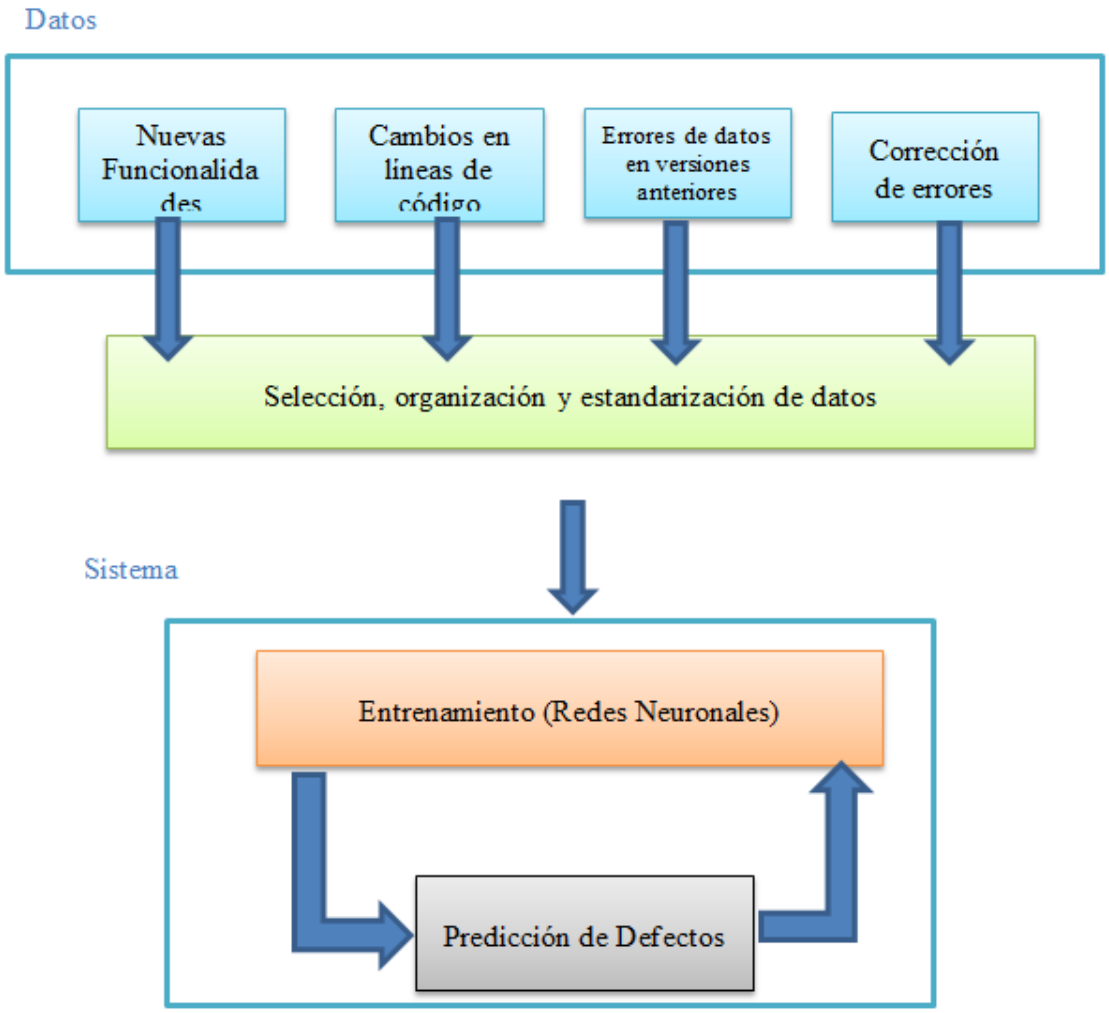


Ilustración 1 Modelo Propuesto

1.9 Resultados Esperados:

La predicción de errores en proyectos de software es de gran importancia hoy en día para las empresas desarrolladoras de aplicaciones. Realizar esta predicción a mediano plazo es una necesidad vital para estas empresas, dado que de ello depende la satisfacción de los usuarios, los cuales se verían perjudicados en caso de que los aplicativos dejen de trabajar. El presente trabajo pretende demostrar que el empleo de técnicas computacionales basadas en inteligencia artificial, como las Redes Neuronales Artificiales, reducen el nivel de error de las predicciones de tendencia a fallos de un software.

2. ANÁLISIS DE ANTECEDENTES

La predicción de defectos en desarrollos de software es un área de investigación relativamente nueva que implica el uso de varias técnicas de inteligencia artificial como la minería de datos ya que contiene la identificación y localización de defectos en los proyectos de software.

La medición de software de manera continua y disciplinada proporciona muchas ventajas tales como la estimación precisa de los costos del proyecto y su duración, así como la mejora del producto y las cualidades del proceso. Este trabajo tiene por objetivo proponer un modelo para predecir el número de defectos en módulo de un proyecto de software que está próximo a entrar en funcionamiento. El modelo a proponer tiene por objeto predecir los defectos introducidos mediante el análisis de los tipos de cambios de una manera objetiva y formal, así como teniendo en cuenta los cambios en las líneas de código (LOC). Los predictores de defectos son herramientas útiles para los directores de proyectos como para los desarrolladores ya que pueden ayudar a reducir los tiempos de prueba. El modelo a proponer puede ayudar a los ingenieros de software en la determinación de la estabilidad del software antes de que entre a funcionamiento.

El conjunto de datos a analizar son obtenidos de los desarrollo hechos por la División de Sistemas de la Universidad Tecnológica de Pereira en los últimos 7 años.

La técnica a utilizar es Redes Neuronales, se desarrollará una herramienta en un lenguaje de programación Delphi y se probará con datos de otros proyectos realizados los cuales están documentados en artículos y se compararan los resultados, luego con la herramienta ya entrenada se utilizaran los datos aportados por la División de Sistemas y se analizarán los resultados.

El objetivo principal de la investigación es predecir defectos en los módulos de software desarrollados. Por lo tanto, la recopilación de datos y una definición clara de los ¨bugs¨ son dos de los factores críticos para el éxito de una predicción. Las listas de control de errores son los las principales fuentes para el análisis, algunos sistemas de control de defectos pueden llevar a un riesgo ya que alguien fácilmente puede ingresar un tipo de error sin ningún tipo de aprobación o de filtro. Por lo tanto, ningún sistema público puede ser considerado como un sistema fiable de seguimiento en el ámbito de esta investigación.

Al seleccionar el sistema de seguimiento de fallos, también es importante que se incluyan las siguientes mediciones: tipo de error, el módulo en el que se observa el error, la clasificación de los errores (interfaz de usuario, la función interfaz, etc), y la gravedad del error. Un error puede provocar un fallo total de un sistema, mientras que otro sólo puede degradar el rendimiento. En contexto, la gravedad del fallo describe el impacto de un error en el sistema de software.

El sistema de predicción propuesto será capaz de predecir no sólo a nivel global, sino también a nivel de módulo.

2.1 Medición de Software

Aunque existen varios métodos formales y herramientas para automatizar un software, la mayor parte de su desarrollo depende de decisiones humanas. Por lo

tanto, los errores conocidos como “bugs” son en su mayoría introducidos como consecuencia del factor humano.

La calidad del software está directamente relacionada con su número de defectos, minimizar este número requiere de una “prueba” completa y precisa. Los modelos de predicción de defectos son herramientas útiles para realizar pruebas a un software, estimaciones precisas de los módulos defectuosos pueden producir disminución de tiempos de prueba, y pueden beneficiar a los administradores de proyectos en términos de asignación de recursos de una forma más eficaz.

Los investigadores han centrado sus investigaciones en modelos de predicción de propensión a fallos (PPF), estos modelos tienden a predecir que partes del software desarrollado recientemente pueden ser más propensos a fallas. De acuerdo a los modelos de PPF, un sistema consta de los componentes descritos por métricas (propiedades numéricas), tales como la complejidad del código o el número de cambios pre-lanzamiento. Algunos modelos estadísticos, como árboles de decisión o regresión logística, utilizan estos parámetros para predecir la tendencia a errores. Inicialmente, se entrena el modelo estadístico con los datos de un software similar para el cual conocemos las métricas de los componentes. Una vez entrenado el modelo se puede utilizar para predecir la propensión del nuevo sistema a los fallos.

2.2 Elementos que complican la predicción

Aplicaciones cada vez más complejas: En los sistemas de información se ha producido una evolución que va desde el uso de información con fines puramente descriptivos, donde la única funcionalidad requerida es el almacenamiento y recuperación de datos, al uso de la misma con fines predictivos y de toma de decisiones, para lo cual la funcionalidad requerida es considerablemente más compleja (Gordillo 1998).

Carencia de datos históricos de proyectos ya terminados: Muchas empresas de desarrollo de software realizan aplicaciones muy completas en su funcionalidad

pero esta completitud dura poco debido a que no hay documentación sobre su creación, es decir, no hay información de código fuente ni de resultados de las pruebas que se le hicieron al proyecto en cada etapa de su desarrollo. Se dice que su completitud dura poco ya que todo software debe evolucionar dado que las necesidades de la empresa para la cual fue hecho cambian y si no hay documentación, estos cambios son casi imposibles o pueden causar más daños que mejoras. También puede no ser un cambio sino un arreglo debido a un error no detectado antes de su implantación si no hay información ni datos de las pruebas realizadas es muy difícil solucionar dicho fallo.

Poco análisis de los datos existentes: Los datos históricos sobre los desarrollos son utilizados solo para correcciones o mejoras del mismo proyecto o módulo, pero no se utilizan para software nuevo. Esto es un desperdicio de información ya que si la información almacenada de proyectos realizados se organizara, se procesara y se analizara mediante una herramienta especializada se podría predecir el estado de nuevos desarrollos y con base en esto decidir si este saldrá a funcionamiento o debe ser reconfigurado.

Técnicas de estimación inexactas: En la actualidad se manejan técnicas de estimación para calcular el costo, duración de desarrollo y mano de obra de un proyecto de software, estas técnicas aunque deficientes dan una idea de los recursos a invertir y el valor del proyecto. Ninguna de esas técnicas ha dado un valor preciso, ya que los proyectos de software siempre se demoran más de lo proyectado y en muchos casos generan pérdidas a la empresa de desarrollo precisamente por su mala planeación.

Diferencias entre los modelos resultantes en cada etapa de desarrollo: Básicamente el desarrollo de software maneja 6 etapas básicas que son: requerimientos, análisis, diseño, desarrollo, pruebas e implantación. En cada etapa del desarrollo se deben dejar las especificaciones necesarias para continuar

con la siguiente, pero en muchos casos el modelo resultante en cada fase no concuerda o presenta inconsistencias, esto hace que los modelos resultantes en cada etapa sean diferentes, en muchos casos lo que se diseña es muy diferente a lo que se programa.

Documentación incompleta e inexacta lo que genera software poco flexible:

En cada etapa del desarrollo debe quedar una documentación detallada de los procesos que se realizaron, debido a que si se requiere un cambio se revisa la documentación señalando en qué parte del código se hace la modificación, ya que un cambio mal hecho puede afectar el rendimiento de todo el sistema. Si esta documentación está incompleta o errónea los cambios son muy riesgosos o el costo es muy alto ya que requiere más tiempo y mano de obra.

Pruebas incompletas: Cuando los proyectos de software se demoran más de lo proyectado las últimas etapas como las de prueba se reducen en tiempo y complejidad, y la prisa por entregar el aplicativo prima sobre lo planeado, esto es gravísimo puesto que las pruebas son casi tan importantes como el desarrollo en sí. Si no hay pruebas adecuadas el software puede presentar fallos en su funcionamiento lo que puede causar pérdidas de tiempo y dinero tanto para la empresa que desarrolló el proyecto como para la empresa que utiliza el aplicativo poniéndose en riesgo la información manejada.

2.3 Técnicas de evaluación y predicción

La Inteligencia Artificial es una de las disciplinas computacionales cuyas técnicas son más demandadas actualmente en diversos entornos, debido a su capacidad para dotar de un comportamiento inteligente a muchas aplicaciones. Así, por ejemplo, la incorporación de agentes de decisión inteligente, redes neuronales, sistemas expertos, algoritmos genéticos, etc. para la optimización de sistemas de

producción es una tendencia activa en el ambiente industrial de países con alto desarrollo tecnológico y con una gran inversión en investigación y desarrollo. Dichos componentes de la Inteligencia Artificial tienen como función principal controlar de manera independiente, y en coordinación con otros agentes, componentes industriales tales como celdas de manufactura o ensamblaje, operaciones de mantenimiento, diagnósticos de sistemas, etc (TIC s.f.).

Una incidencia importante de este tipo de técnicas en los procesos productivos de la industria a nivel mundial es el diseño de sistemas de soporte para la toma de decisiones. Además, la mayoría de los sistemas de inteligencia artificial, tienen la peculiaridad de “aprender”, lo que les permite ir perfeccionando su tarea conforme pasa el tiempo, usando para ello los ejemplos o casos con los que tratan. Finalmente, otro campo importante de aplicación es la propia Informática, en donde se puede realizar el diagnóstico de fallos, la detección de intrusiones en redes de ordenadores (TIC s.f.).

2.4 Antecedentes de la investigación

La obtención de conocimiento a partir de un volumen de datos, tiene como fin descubrir patrones válidos, comprensibles y útiles. Los seres humanos tienen una capacidad de reconocer patrones a su alrededor de acuerdo a su conocimiento previo. Las Redes Neuronales han querido emular estas capacidades de reconocimiento para deducir y predecir información valiosa. Los proyectos e investigaciones relacionados con la Redes Neuronales, técnicas de predicción y Minería de Datos han contribuido enormemente al desarrollo, mejoramiento y especialización de los algoritmos y metodologías de extracción de conocimiento, obteniendo cada vez con mayor precisión, información significativa para una organización o empresa.

La Minería de Datos, apoya la toma de decisiones en una organización, facilitando la administración del negocio. También es de gran ayuda en el ámbito académico,

permitiendo abordar los problemas desde un punto de vista apropiado para resolver problemas desde su forma más básica.

Titulo: **A defect prediction method for software versioning.**

Autores: Yomi Kastro Æ Ays¸e Basar Bener.

Base de Datos: Springer.

URL: http://www.springerlink.com/content/e231k59u672478l3/

Idioma: Ingles

Fecha: 2008

Resumen:

New methodologies and tools have gradually made the life cycle for software development more human-independent. Much of the research in this field focuses on defect reduction, defect identification and defect prediction. Defect prediction is a relatively new research area that involves using various methods from artificial intelligence to data mining. Identifying and locating defects in software projects is a difficult task.

Measuring software in a continuous and disciplined manner provides many advantages such as the accurate estimation of project costs and schedules as well as improving product and process qualities. This study aims to propose a model to predict the number of defects in the new version of a software product with respect to the previous stable version. The new version may contain changes related to a new feature or a modification in the algorithm or bug fixes. Our proposed model aims to predict the new defects introduced into the new version by analyzing the types of changes in an objective and formal manner as well as considering the lines of code (LOC) change. Defect predictors are helpful tools for both project managers and developers. Accurate predictors may help reducing test times and guide developers towards implementing higher quality codes.

Our proposed model can aid software engineers in determining the stability of software before it goes on production. Furthermore, such a model may provide

useful insight for understanding the effects of a feature, bug fix or change in the process of defect detection.

Comentario

En este artículo se propone un modelo de predicción de defectos para diferentes versiones de software y se recogen los datos necesarios para llevar a cabo varios experimentos. La investigación tiene tres aportes principales: Datos, el método y el experimento. La primera contribución es la colección y preprocesamiento de gran cantidad de datos "bugs" y cambios en dos versiones diferentes de un desarrollo de software. Los datos contienen importante información estadística relativa al ciclo de vida en el desarrollo de versiones de software. La segunda contribución de esta investigación ha sido proponer un nuevo método para la predicción de defectos sistemas. La introducción del concepto de sistemas de predicción de defectos de software de versiones y diferencia entre las versiones con temas importantes fueron cubiertos a través de esta investigación. Tener dos diferentes experimentos aumentó la validez y utilidad de la propuesta de modelo. También hay contribuciones prácticas de esta investigación a las empresas de desarrollo de software.

Con una colección de métricas bien definidas, se puede usar el modelo propuesto como un mecanismo de control de versiones para las nuevas versiones de software. El modelo que es propuesto podría ayudar de forma proactiva la detección de defectos, y como resultado, adquirir un mejor criterio en la asignación de sus recursos al momento de desarrollar nuevas aplicaciones.

Titulo: Integrating in-process software defect prediction with association mining to discover defect pattern.

Autores: Ching-Pao Chang, Chih-Ping Chu, Yu-Fang Yeh.

Base de Datos: Proquest.

URL:

http://proquest.umi.com/pqdweb?index=0&did=1610379611&SrchMode=1&sid=1&

Fmt=2&VInst=PROD&VType=PQD&RQT=309&VName=PQD&TS=1300236088&clientId=47007

Idioma: Ingles

Fecha: 2008

Resumen:

Rather than detecting defects at an early stage to reduce their impact, defect prevention means that defects are prevented from occurring in advance. Causal analysis is a common approach to discover the causes of defects and take corrective actions. However, selecting defects to analyze among large amounts of reported defects is time consuming, and requires significant effort. To address this problem, this study proposes a defect prediction approach where the reported defects and performed actions are utilized to discover the patterns of actions which are likely to cause defects. The approach proposed in this study is adapted from the Action-Based Defect Prediction (ABDP), an approach uses the classification with decision tree technique to build a prediction model, and performs association rule mining on the records of actions and defects. An action is defined as a basic operation used to perform a software project, while a defect is defined as software flaws and can arise at any stage of the software process. The association rule mining finds the maximum rule set with specific minimum support and confidence and thus the discovered knowledge can be utilized to interpret the prediction models and software process behaviors. The discovered patterns then can be applied to predict the defects generated by the subsequent actions and take necessary corrective actions to avoid defects.

The proposed defect prediction approach applies association rule mining to discover defect patterns, and multi-interval discretization to handle the continuous attributes of actions. The proposed approach is applied to a business project, giving excellent prediction results and revealing the efficiency of the proposed approach. The main benefit of using this approach is that the discovered defect patterns can be used to evaluate subsequent actions for in-process projects, and

reduce variance of the reported data resulting from different projects. Additionally, the discovered patterns can be used in causal analysis to identify the causes of defects for software process improvement.

Comentario

La predicción defectos es una actividad importante para mejorar los procesos de software. Este estudio presenta Predicción de Defectos Basado en Reglas de Asociación, las cuales están relacionadas a la técnica de minería de datos. El enfoque propuesto se puede aplicar al proceso de desarrollo de software para predecir las acciones que pueden causar defectos altos. La principal ventaja es la predicción en proceso, en la que los datos de entrenamiento se pueden obtener del proyecto en ejecución para la construcción del conjunto de reglas. La prevención de defectos se centra principalmente en mejorar la calidad del software y la productividad en una organización, en las que el análisis causal es la principal actividad para identificar las causas de la aparición de defectos.
En el proyecto el conjunto de datos utilizados para extraer las reglas se obtendrá de los informes de las operaciones y los defectos de proyectos anteriores. Los resultados de la predicción pueden ser aplicados en el proceso de análisis causal y la planificación de acciones correctivas para prevenir que estos defectos se produzcan.

Titulo: Aplicación De Técnicas De Minería De Datos En La Construcción Y Validación De Modelos Predictivos Y Asociativos A Partir De Especificaciones De Requisitos.
Autores: María N. Moreno García, Luis A. Miguel Quintales, Francisco J. García Peñalvo y M. José Polo Martín.
URL:http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-84/paper4.pdf
Idioma: Español
Fecha: 2002

Resumen:

La medición del software está adquiriendo una gran importancia debido a que cada vez se hace más patente la necesidad de obtener datos objetivos que permitan evaluar, predecir y mejorar la calidad del software así como el tiempo y coste de desarrollo del mismo.

El valor de las mediciones aumenta cuando se realiza sobre modelos construidos en las primeras fases del proyecto ya que los resultados obtenidos permiten tenerlo bajo control en todo momento y corregir a tiempo posibles desviaciones. La proliferación actual de métricas y el gran volumen de datos que se maneja ha puesto de manifiesto que las técnicas clásicas de análisis de datos son insuficientes para lograr los objetivos perseguidos. En este trabajo se presenta la forma en que pueden aplicarse las nuevas técnicas de minería de datos en la construcción y validación de modelos de ingeniería del software, cambiando el análisis tradicional de datos dirigido a la verificación por un enfoque de análisis de datos dirigido al descubrimiento del conocimiento.

Las primeras etapas del desarrollo de software son cruciales en la consecución de productos de calidad dentro de los límites de tiempo y coste establecidos para un proyecto. Los errores introducidos en dichas etapas o durante su evolución son causa frecuente de dificultades en el mantenimiento, baja reutilización y comportamiento defectuoso de los programas. Igualmente, las malas estimaciones realizadas al comienzo del proyecto tienen consecuencias desastrosas en cuanto a costes y plazos de entrega. Estas son las principales causas por las que la medición del software en el ámbito de la especificación de requisitos (ERS) está adquiriendo cada vez mayor importancia, debido a la necesidad de obtener datos objetivos que contribuyan a mejorar la calidad desde las primeras etapas del proyecto.

Los estudios relacionados con la medición en el nivel de la especificación de requisitos se han centrado fundamentalmente en el desarrollo de métricas para determinar el tamaño y la funcionalidad del software. Entre las de mayor difusión se encuentran las métricas de puntos de función (Albrecht 1979), métricas Bang

(DeMarco 1982) o los puntos objeto (Boehm, Clark y al. 1995). La medición de atributos de calidad de las especificaciones de requisitos del software (ERS) ha sido también objeto de algunos trabajos que van desde la medición de especificaciones formales (Samson y Dugard 1990) hasta la aplicación de métricas para evaluar la calidad de especificaciones expresadas informalmente en lenguaje natural (métricas de facilidad de comprensión del texto contenido en los documentos (Lehner 1993) o métricas de estructura y organización en documentos convencionales (Stevens 1989) y con hipertexto (French y Powell 1997) (Roth y Hobbs 1994)), pasando por técnicas encaminadas a determinar el cumplimiento de los estándares, directrices, especificaciones y procedimientos, que requieren información procedente de revisiones técnicas, inspecciones**,** *Walkthrough*, o auditorías (al. 1993) (Brykczynskki 1999) (Farbey 1990). La creciente adopción de la tecnología de orientación a objetos en el desarrollo de software ha dado lugar a la aparición de nuevas métricas específicas para este tipo de sistemas (Kemerer 1994), (Shepperd 1995). Recientemente se han propuesto métricas para la evaluación de la calidad a partir de modelos producidos en etapas iniciales del ciclo de vida, como son las métricas de calidad y complejidad en modelos OMT (Genero, Manso y García, Assessing the quality and the complexity of OMT models 1999), métricas de calidad de los diagramas de clases en UML (Genero y M. Piattini, Una propuesta para medir la calidad de los diagramas de clases en UML 2000) o las técnicas de medición de modelos conceptuales basados en eventos (Poels, On the measurements of event-based object-oriented conceptual models 2000).

La proliferación actual de métricas y la necesidad de medir diferentes aspectos del software está contribuyendo a crear confusión sobre las relaciones entre tales medidas, así como sobre su forma y ámbito de aplicación. Este hecho ha abierto una nueva vía en la investigación orientada hacia la propuesta de modelos, arquitecturas y marcos de referencia (.frameworks.) que permitan la organización de las medidas y la clasificación de las entidades de software susceptibles de medir (Moreno, y otros 2001) (Poels, Towards a size measurement framework for

object-oriented especifications 1998) (Briand y Wüst 1999). La construcción de modelos sobre diferentes aspectos del software requiere la recolección de numerosos datos procedentes de observaciones empíricas. Los avances tecnológicos actuales posibilitan la rápida obtención de grandes cantidades de datos de fuentes muy diversas, así como el almacenamiento eficiente de los mismos. Dichos datos encierran información muy valiosa que puede tratarse mediante los métodos tradicionales de análisis de datos, sin embargo estos métodos no son capaces de encontrar toda la información útil latente en la gran masa de datos que se maneja. En ese contexto, las técnicas de minería de datos surgen como las mejores herramientas para realizar exploraciones más profundas y extraer información nueva, útil y no trivial que se encuentra oculta en grandes volúmenes de datos.

En este trabajo se muestra la aplicación práctica de técnicas de minería de datos en la construcción y validación de modelos de ingeniería del software que relacionan diferentes atributos de la ERS con el objeto de predecir características del producto final y descubrir patrones y afinidades ocultas entre dichos atributos.

Comentario

En el apartado anterior se han ofrecido únicamente unos pocos ejemplos del uso de técnicas de minería de datos en la construcción de modelos predictivos y asociativos con datos procedentes de mediciones realizadas en el ámbito de la especificación de requisitos, no obstante las posibilidades que ofrece este nuevo enfoque de tratamiento de datos son mucho mayores, ya que el número de técnicas que engloba es mucho más amplio. Por otra parte, los métodos de minería de datos llevan asociados una serie de mecanismos que permiten realizar una mejor validación empírica de los modelos y un análisis de resultados más completo y fiable que el que ofrece el enfoque clásico.

Se pretende proponer un modelo de predicción de defectos para múltiples versiones de software y recoger los datos necesarios para llevar a cabo varios experimentos. La cantidad de datos recogidos da amplia idea sobre los errores y

cambios respecto a las versiones de dos productos de software. Los datos contienen información estadística importante con respecto a los ciclos de vida de desarrollo de versiones de software.

Título: Estimación de variables en proyectos de desarrollo de software (pds).
Autores: Javier Aroba Páez, Luis Isabel Ramos Román, Jose C. Riquelme Santos.
URL:
https://www.google.com.co/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CBsQFjAAahUKEwjJi83XxujIAhVDHx4KHfMCDXI&url=http%3A%2F%2Fwww.aemes.org%2Findex.php%2Frevista-de-procesos-y-metricas%2Fnumeros-publicados%2Fano-2004-volumen-1%2Fvolumen-1-g-numero-2-g-agosto-2004%2F146-estimacion-de-variables-en-proyectos-de-desarrollo-de-software-pds%2Fdownload&usg=AFQjCNHIo8ZRhuEqBTwhr_bYaTQ1v55WHg&sig2=rTKSSitk9YZbpkb5kMG2uQ&bvm=bv.106130839,d.dmo
Idioma: Español
Fecha: Agosto 2004

Resumen: La aplicación de determinadas técnicas de Data Mining sobre bases de datos numéricas de proyectos de desarrollo de software (PDS), permite, entre otras cosas, obtener información cualitativa sobre la evolución del proyecto. Muchas de estas técnicas son descriptivas, como por ejemplo el clustering, por lo que no se tiene a priori capacidad de predicción de resultados (variables del proyecto) a partir de nuevos datos (atributos del proyecto) de PDS. Para estimar estas variables a partir de nuevos valores en los atributos, se proponen en esta investigación diversas técnicas, de forma que se puedan comparar los resultados obtenidos. El objetivo es comprobar si se pueden estimar las variables de un proyecto, a partir de una nueva serie de valores de atributos, sin tener que simular todo el proyecto, y con unos márgenes de error bajos.

Comentarios:

El proceso de estimación del software es un área en el que se están proponiendo métodos y técnicas desde hace 15 años. La mayoría de los procesos existentes describen cómo aplicar un método simple de predicción, que normalmente está basado en uno o más modelos algorítmicos.

En líneas generales, estimar consiste en determinar el valor de una variable desconocida a partir de otras conocidas, o de una pequeña cantidad de valores conocidos de esa misma variable. La estimación forma parte de la inferencia estadística.

Si enfocamos todos los conceptos estadísticos referentes a estimación (estadístico de una muestra, parámetro, estimador, población, muestra) hacia el caso particular de la estimación del software, tendremos que los parámetros a estimar son: el tamaño del proyecto, el esfuerzo para realizar el mismo, el costo y el tiempo que se tardará en desarrollarlo.

Titulo: Combining neural networks and genetic algorithms for predicting the reliability of repairable systems.

Autores: Yi-Hui Liang.

Base de Datos: Proquest.

URL:

http://search.proquest.com.ezproxy.utp.edu.co/docview/197669297/fulltextPDF/12E30E7EED87382570A/16?accountid=45809

Idioma: Ingles

Fecha: 2007

Resumen: The purpose of this paper is to propose an accurate product reliability prediction model in order to enhance product quality and reduce product costs. This study proposes a new method for predicting the reliability of repairable systems. The novel method employed constructs a predictive model by integrating neural networks and genetic algorithms. The novel method employed constructs a predictive model by integrating neural networks and genetic algorithms. Genetic

algorithms are used to globally optimize the number of neurons in the hidden layer, the learning rate and momentum of neural network architecture. This study only adopts real failure data from an electronic system to verify the feasibility and effectiveness of the proposed method. Future research may use other product's failure data to verify the proposed method. The proposed method is superior to ARIMA and neural network model prediction techniques in the reliability of repairable systems. Based on the more accurate analytical results achieved by the proposed method, engineers or management authorities can take follow-up actions to ensure that products meet quality requirements, provide logistical support and correct product design.

Comentario

Un sistema reparable es un sistema que, después de no poder realizar una o más de sus funciones de manera satisfactoria, se puede restaurar a un rendimiento óptimo sin tener que reemplazar todo el sistema. General mecánico, eléctrico y de comunicaciones, son ejemplos de sistemas reparables.

Las redes neuronales artificiales son capaces de aprender la mecánica de la serie de tiempo, es difícil decidir qué datos de entrada, estructura de la red y los parámetros de aprender a utilizar. Sin embargo, los algoritmos genéticos se pueden aplicar como una búsqueda de optimización para determinar el óptimo en el diseño de la estructura de red neuronal de maximización combinación de entrada de datos, optimización de la estructura de la red, la tasa de aprendizaje y de optimización dinámica.

Debido a las respectivas ventajas del modelo de redes neuronales y algoritmos genéticos, este estudio tiene como objetivo principal de combinar el modelo de redes neuronales y algoritmos genéticos para desarrollar un nuevo medio de forma efectiva la predicción de la fiabilidad de sistemas reparables. Los algoritmos genéticos se utilizan para optimizar todo el mundo el número de neuronas en la capa oculta, la tasa de aprendizaje y el impulso de la arquitectura de red neuronal. Los resultados de este estudio son capaces de proporcionar una valiosa

herramienta de referencia para los ingenieros en la construcción de sistemas de calidad de retroalimentación para evaluar las condiciones actuales de calidad, la prestación de apoyo logístico, para corregir el diseño de productos.

Titulo: **Mining Software History to Improve Software Maintenance Quality: A Case Study.**

Autores: Alexander Tarvo

URL:

http://search.proquest.com.ezproxy.utp.edu.co/docview/215838638/12E2F5E3B5C5BA6B272/1?accountid=45809

Idioma: Ingles

Base de Datos: Proquest

Fecha: 2002

Resumen: Software maintenance includes correcting discovered faults, adapting the system to changes in the environment, and improving the system's reliability and performance.

These activities result in system modifications that are distributed to customers as updates. Microsoft Windows is no exception to this process. After Microsoft releases a new version of Windows, the Windows Serviceability team makes postrelease changes (fixes) to the system. However, any incorrect changes will cause *software regressions*—failures in already stable features and parts of the system.

Regressions are exceptionally painful for customers: imagine what will happen if your computer suddenly stops working after you install an update.

The best method to avoid regressions' negative consequences is extensive testing of all fixes.

However, because of the large user base and the growing number of Windows versions we must support, this method results in a constantly increasing amount of testing for the fixes. This requires us to distribute our limited resources in the most

efficient way: we must detect the riskiest software updates and concentrate on testing them.

This will let us detect and fix possible regressions early, before releasing the updates to customers.

So, we need a tool or a method that can predict the amount of risk for each fix.

To meet this need, I developed the Binary Change Tracer (BCT). BCT extracts information on all changes that have happened to Microsoft Windows. Data mined with this tool let us build a statistical model that predicts each fix's risk of regression. This approach exploits features of a standard engineering process, so it can be used for a variety of software projects.

Comentario

Existe una amplia investigación sobre la predicción del riesgo en proyectos de software. En particular, los investigadores se han concentrado en la predicción de propensión a fallos (PPF). Estos modelos tienden a predecir que partes del sistema de software desarrollado recientemente puede ser más propensos a fallas. De acuerdo a los modelos de PPF, un sistema consta de los componentes descritos por métricas (propiedades numéricas), tales como la complejidad del código y el número de cambios pre-lanzamiento.

Los modelos de PPF no están diseñados para predecir el riesgo de una actualización de software en particular. Sin embargo, se puede construir un modelo de predicción de software que funcione de manera similar a los modelos de PPF: un conjunto de indicadores que describen cada cambio de código.

Título: **Minería de Datos como soporte a la toma de decisiones empresariales.**

Autores: Yelitza Josefina Marcano Anular. Rosalba Talavera Pereira.

URL: http://www.redalyc.org/articulo.oa?id=31005208

Idioma: español

Fecha: 11 de marzo de 2007

Resumen: La tarea por mejorar el acceso a la información está cobrando cada vez más fuerza, especialmente en los negocios actuales, donde se requiere principalmente de procesos basados en el recurso información, de manera automatizada y reutilizable. En ese orden de ideas, este artículo constituye una primera aproximación al área de la Minería de Datos y tiene como objetivo examinar y describir las técnicas y herramientas que emergen en esa área de investigación, apoyándose para ello en una reflexión teórica-cualitativa que contribuya a un mayor entendimiento del alcance y limitaciones de la Minería de Datos como soporte a la toma de decisiones empresariales. Entre los beneficios que ofrece la técnica están la posibilidad de elevar los niveles de competencia de los negocios, basándose en la rapidez para identificar, procesar y extraer la información que realmente es importante, descubriendo conocimiento y patrones en bases de datos. Su facilidad de uso hace que se pueda aplicar a cualquier área del conocimiento. Como limitaciones destacan la necesidad de dedicar mucho esfuerzo al establecimiento de medidas de evaluación del resultado derivado de la aplicación de la minería, así como el desafío que representa analizar datos que cambian en tiempo real.

En la actualidad las organizaciones suelen moverse dentro de estructuras identificadas con un cambio continuo; por ello, las empresas privadas tanto como las públicas deben tener la capacidad de ser adaptativas, aprender cómo resolver problemas y generar conocimiento, para establecer nuevos métodos en pro de la resolución de los mismos.

Las organizaciones, en la búsqueda por la obtención de los mejores resultados de su gestión organizacional, adoptan la flexibilización como estrategia, con el objetivo de adecuarse a un mercado globalizado, dando origen a un proceso que incide en su sistema estructural. Así pues, una empresa flexible es la que se orienta hacia los clientes, posee tecnología nueva y presenta acuerdos laterales de organización e innovación (Hansen y Mouritsen 1999).

Las aplicaciones necesarias para gestionar el flujo de información en las actividades de negocio se pueden clasificar en dos importantes categorías: las aplicaciones que manejan las transacciones y las estadísticas que ayudan a convertir los datos en información útil para la toma de decisiones. Además está el sistema de indicadores, formado por las bases de datos donde se almacenan los datos importantes para evaluar y mejorar el funcionamiento de las actividades que componen la cadena de suministro y por aplicaciones de análisis que facilitan la comprensión de las tendencias y patrones presentes en los datos. El sistema de indicadores se considera como un instrumento de integración básico a través de la comunicación y diálogo que se establece, en base a los datos, entre los diferentes actores del proceso.

En la visión de Castañeda y Rodríguez (2003), el uso de la Minería de Datos o Data Mining, como soporte a las decisiones en las actividades de negocio, requiere mucho más que la aplicación de sofisticadas técnicas como redes neuronales o árboles de decisión sobre las tablas de datos. Por esta razón, en el presente documento se muestra a la Minería de Datos por un lado, como uno de los pasos del proceso de descubrimiento de conocimiento en base de datos (KDD) y por otro lado como un proceso que consta de diferentes fases, en las cuales se utilizan como apoyo, técnicas relacionadas con la estadística, el reconocimiento de patrones y algoritmos de aprendizaje, entre otras.

Todos estos estudios han incrementado el deseo desenfrenado por demandar un mayor control de los procesos u operaciones y servicios, visto como núcleo de una gestión global, fundamental para proporcionar servicios de calidad y lograr un rendimiento óptimo de las inversiones, en infraestructuras comerciales, en un entorno competitivo dirigido hacia una gestión de clientes.

Este trabajo constituye un primer acercamiento a un área de investigación de reciente data, el cual tiene como propósito presentar algunas bases teóricas sobre la incidencia de la Minería de Datos como soporte en la toma de decisiones, aplicadas a las actividades de negocio. La elaboración de la reflexión teórica hace énfasis en los postulados metodológicos del paradigma cualitativo, el cual permite

la construcción del conocimiento partiendo de una visión integral, interpretativa y contextual del fenómeno a estudiar. Las teorías consultadas se interpretaron para establecer por inferencia deductiva algunas consideraciones relacionadas a la Minería de Datos y a algunos indicadores que permitan medir el interés y el impacto del conocimiento que se puede obtener, al emplearla, como soporte para la toma de decisiones en las organizaciones.

Comentarios:

La información ya no se percibe como un activo. Millones de transacciones de negocios son registradas en los almacenes de datos todos los días. La adquisición, almacenamiento y gestión de información son comunes y menudo automatizadas. El reto actualmente es ser capaz de utilizar dicha información, para obtener una mejor comprensión del pasado y predecir o influir en el futuro a través de una mejor toma de decisiones. Las tecnologías de minería de datos y los sistemas de soporte de decisiones están respondiendo a este desafío.

En términos generales los datos son el corazón de muchos procesos de negocio y la minería de datos es la ciencia de la transformación de datos en conocimiento, conocimiento en acción, y la acción en valor.

Titulo: Impact Analysis of the Inspection Process for Effective Defect Management in Software Development.

Autores: Nair, T R Gopalakrishnan.

Base de Datos: Proquest

URL:

http://search.proquest.com.ezproxy.utp.edu.co/docview/214067480/fulltext/12E2F6BA928799F3385/1?accountid=45809

Idioma: Ingles

Fecha: 2010

Resumen: A key challenge of the software industry is to engineer software products with minimal post-deployment defects. The two successful approaches of

managing defects are defect detection and defect prevention. Inspection has proven to be the most mature, valuable, and competent technique in this challenging area. An awareness of the effective implementation of inspection activity serves to be one of the best practices in the software development process. This article is an empirical analysis of various projects across several service-based and product-based software organizations. The study indicates the impact of several parameters on the effectiveness of inspection activity. These parameters include the number of inspectors, inspection time, preparation time, experience level, and skill of inspectors. An empirical analysis of the data enables one to introduce a range of DI values chart and variation-analysis chart. This article introduces a process metric-depth of inspection (DI-and a people metric-inspection performance metric (IPM)-to quantify the effectiveness of the inspection process and the people enabling the process. The study suggests the desirable ranges of DI and IPM, and the experience level of inspectors to achieve a 90-percent or greater rate of defect-free product. The introduction of these metrics reflects a continual process of improvement and success level of the organization. These metrics will further pave the way for the stakeholders to have a deep visibility into the process and thereby justify the developmental cost. The article further recommends that management within the software community emphasize environmental conditions and motivational factors for enhancing the skills of inspectors.

Comentarios:

Un control eficaz de los defectos es una de las actividades más importantes en la industria del software. La toma de conciencia de los métodos de detección de defectos permite la prevención de defectos, y la prevención de defectos es un paso esencial hacia el logro de madurez de los procesos. La inspección sigue demostrando ser la técnica más eficaz y eficiente para la detección y prevención de defectos. La mejora continua del proceso, sin embargo, exige una necesidad

de medir el impacto del proceso de inspección en la entrega de producto de alta calidad.

Este trabajo es un estudio empírico en varios proyectos a través de varias industrias de software basadas en servicios y en base a productos para identificar la eficacia del proceso de inspección. El número de inspectores, el tiempo de preparación, la experiencia del tiempo de inspección, y la habilidad de los inspectores son los principales parámetros que influyen en la eficacia de las actividades de inspección. Un análisis más profundo de varios proyectos y una comparación entre los proyectos similares indican la necesidad de cuantificar el rendimiento de la inspección.

3. MARCO TEÓRICO

Este capítulo tiene como objetivo mostrar un conjunto de conocimientos que permita delimitar teóricamente los conceptos planteados, es decir, es una delimitación teórica del problema de investigación en lo relacionado a las Redes Neuronales.

3.1 Inteligencia Artificial

Desde hace 5 décadas, se han propuesto muchas definiciones de lo que es la Inteligencia Artificial, las cuales en ningún caso han logrado la aceptación general de toda la comunidad científica relacionada con el área. Esto se debe a que el concepto de inteligencia no está claramente definido y sólo se ha relacionado con los recuerdos, pensamientos y actos de los seres humanos. También podría deberse a que muchas personas no aceptan siquiera la idea de que una máquina pueda superarlos no solo en el campo laboral sino también en el campo intelectual, por eso cuando se empieza a hablar sobre los grandes adelantos que en el campo de la IA se están dando, algunos seres humanos tienden a desprestigiar dicha información o subvaloran el beneficio o aporte científico que el invento puede ofrecer (Arias y Cardona 1997).

Experimentos recientes hechos con chimpancés han demostrado que ellos tienen la habilidad para reconocer objetos y que además tienen la capacidad para elegir según sus gustos ante una situación particular. El experimento consistió en que a un chimpancé se le mostraron imágenes por video de unas hembras de su especie, hasta que el chimpancé demostró interés por una de ellas, luego fue llevado a un lugar donde se encontraban todas las hembras mostradas en el video y el chimpancé reconoció a aquella que le había interesado. Los seres humanos nos asombramos ante esta habilidad demostrada por un chimpancé y decimos que esto es una prueba segura de que en ellos hay cierto nivel de inteligencia, sin embargo, la mayoría de la gente seguramente no diría que un robot es inteligente si este tiene la misma habilidad (Arias y Cardona 1997).

Pero la IA no tiene por objetivo crear sistemas para suplantar al hombre, lo que ella busca es colaborarle con aquellas tareas las cuales requieren de mucha precisión, rapidez y efectividad. Tal es el caso de la NASA, la cual envía sondas al espacio para fines investigativos, y en muchos de estos casos estas se han perdido en otros planetas del sistema solar o terminan vagando por el espacio ya que sus acciones se encuentran tan mecanizadas que cuando resulta un imprevisto, no saben que acción emprender y la misión por lo general fracasa. La IA puede dar solución a este tipo de problema con la creación de máquinas capaces de tomar decisiones por sí solas, ante una situación inesperada. Otra aplicación de IA podría ser en el campo de la seguridad ya que pueden existir sistemas de seguridad que no solo detecten el peligro sino que tomen decisiones y actúen rápidamente sin necesidad de esperar la intervención humana, tal es el caso de un banco en el cual podría instalarse un sistema que analizara a las personas que ingresan; revisara sus movimientos, su temperatura corporal, su pulso o hasta su tono de voz con el objetivo de detectar algún comportamiento extraño que podría significar peligro para las demás personas que están en el banco, si este peligro es detectado el sistema podría dar la orden para que se vigile a esta persona pues podría ser un posible delincuente, claro está sin que

ninguna de las personas se dé cuenta, esto es solo como una medida preventiva (Arias y Cardona 1997).

Uno de los objetivos que busca la IA es el de entender y explicar el pensamiento y la conducta humana desde un punto de vista plenamente científico, inclusive trata de comprender aquellos fenómenos que no se consideran propios de la conducta ordinaria, como mostrar destreza y agilidad ante una situación de peligro o la capacidad de decisión ante un negocio importante (Arias y Cardona 1997).

3.1.1 ¿Qué es la Inteligencia Artificial?

Existen muchas definiciones que pretenden explicar la IA, pero como se mencionó anteriormente no hay un consenso general cuya definición satisfaga a todos los investigadores, pero podríamos darle una explicación simple que permita dar al lector una idea de lo que esta ciencia pretende (Arias y Cardona 1997).

"La Inteligencia Artificial es el arte de diseñar sistemas capaces de emular el pensamiento humano".*

Para una correcta imitación de éste, el sistema tiene que disponer de un conocimiento amplio del campo en el que se va a desempeñar, ya sea el de la medicina, astronomía o la seguridad; aunque también debería tener un conocimiento general acerca de la cultura humana, es decir, su historia, los diferentes estilos de vida, hasta los pasatiempos más populares; esto con el fin de hacer al sistema más amigable. Debe tener la habilidad de interactuar con otros sistemas y con los seres humanos de forma lógica y en un lenguaje comprensible que le permita comunicar sus ideas y a su vez interpretar y aprender con las respuestas o mensajes que le sean enviados. La percepción visual del entorno para conocer e interpretar el medio en el cual el sistema vive y se desarrolla debe estar disponible y sin limitaciones. Debe tomar decisiones de una manera

*Definición desarrollada por la autora.

autónoma y flexible, es decir, que ante dos situaciones un poco similares, el sistema ha de tomar la decisión que considere más apropiada para resolver cada una según el conocimiento que el sistema tenga y la prioridad que este le asigno a cada uno de las situaciones a resolver. La creación de equipos capaces de adquirir conocimiento por medio de la experiencia es tal vez una de las prioridades más altas de la IA; la experiencia es obtenida gracias al dominio por parte del sistema de las características mencionadas anteriormente; que aunque para los humanos realizarlas es algo trivial y no afecta en nada el desempeño de sus tareas diarias, para un sistema creado por la IA constituye una de las principales barreras para su creación (Arias y Cardona 1997).

Existen grandes discrepancias en cómo diseñar el proceso de razonamiento del sistema que se va a crear, algunos piensan que con la ayuda de la ciencia cognoscitiva la IA puede obtener un conocimiento amplio de la mente humana, ya que ofrece caminos y estructuras de cómo funcionan los procesos de razonamiento, lo que permitiría expresar dichos datos en algún programa de computadora y así desarrollar sistemas que simulen con éxito el pensamiento humano. Otros piensan que un sistema inteligente es aquel que resuelve problemas correctamente y cuyos procesos utilizan mecanismos de razonamiento basados en lógica. Pero el uso de inferencias para resolver una situación no basta, ya que no todos los problemas o decisiones de la vida real se resuelven con deducciones correctas, hay decisiones que son las correctas pero que no resultan del razonamiento lógico, por ejemplo cuando ponemos nuestra mano en una hornilla caliente, inmediatamente la quitamos; esto fue la decisión correcta pero no se obtuvo mediante inferencias, a esto le llamamos acto reflejo (Arias y Cardona 1997).

Actuar racionalmente significa actuar de modo que se logren los objetivos deseados, teniendo en cuenta que no siempre el proceso de razonamiento se

hace por medio de deducciones lógicas, este es el principio de los Agentes Racionales. Es muy importante la información proporcionada por la ciencia cognoscitiva, ya que como se dijo anteriormente esta puede proporcionar una representación del conocimiento para poder diseñar agentes racionales con base en esta información. Podemos decir que lo que busca un Agente Racional es lograr sus objetivos, utilizando técnicas de razonamiento que le permitan actuar de acuerdo a la situación, aprender constantemente y adaptarse a las modificaciones que se le presenten en su ambiente (Arias y Cardona 1997).

3.1.2 Disciplinas Relacionadas con la IA

Podríamos decir que la IA se convirtió en una ciencia con la ayuda de otras ciencias. La Inteligencia Artificial ha obtenido teorías, modelos, conclusiones y todo tipo de información de disciplinas como la filosofía, las matemáticas, psicología, lingüística y la ingeniería computacional, que le han permitido desarrollar sistemas realmente asombrosos (Arias y Cardona 1997).

Para la filosofía el pensamiento es un proceso por el cual se refleja o representa la realidad; el hombre compara lo pensado con la realidad para descubrir las relaciones existentes entre los objetos, para aclarar, fundamentar y diferenciar los pensamientos verdaderos de los falsos. Las reflexiones sobre el conocimiento se inician con Platón y Aristóteles en el desarrollo de su doctrina; por ejemplo, distinguen entre el conocimiento sensible y el inteligible, pero no plantean propiamente ninguna duda en torno al conocimiento, también con el desarrollo de los silogismos para explicar el razonamiento deductivo (Arias y Cardona 1997).

Durante la edad media se continúa la tradición Aristotélica; con Bacon en el renacimiento, empiezan a manifestarse inquietudes acerca del origen del conocimiento, sin embargo es en la época moderna a partir de Descartes, cuando se afronta de manera directa la esencia y origen del conocimiento. Los

escolásticos sostenían que “nada hay en el entendimiento que no haya pasado por los sentidos”, es decir, que la mayoría de las ideas tienen su base en la percepción sensible. Los filósofos modernos son menos radicales en esta afirmación, pero si conceden gran importancia al fundamento dado por los sentidos para la abstracción de las ideas, sin dejar de reconocer que hay ideas que no tienen una imagen sensible previa. Posteriormente, diferentes filósofos han dado su aporte a la temática del conocer humano. La filosofía le ha dado a la IA una idea de lo que es el pensamiento humano y cómo funciona el proceso de razonamiento y aprendizaje (Arias y Cardona 1997).

En las **matemáticas**, han surgido teorías formales relacionadas con la lógica, la probabilidad, teoría de decisiones y la computación. La lógica estudia las leyes y reglas del pensamiento para asegurar validez de los razonamientos; dichas reglas poseen un amplio campo de significación por ser el resultado de la abstracción de pensamientos diferentes, particulares y concretos. Las técnicas probabilísticas permiten medir el grado de validez de un razonamiento y con la teoría de decisiones se permite diferenciar las buenas acciones de las malas. Las matemáticas proveyeron las herramientas para manipular las aseveraciones de certeza lógica así como las inciertas de tipo probabilista. Así mismo prepararon el terreno para el manejo del razonamiento con algoritmos (Arias y Cardona 1997).

La **psicología** ofrece herramientas que permiten la investigación de la mente humana, así como un lenguaje científico para expresar las teorías que se van obteniendo. Los psicólogos reforzaron la idea de que los humanos y otros animales podían ser considerados como máquinas para el procesamiento de información (Berthold y Hand 2003).

La **lingüística** ofrece teorías sobre la estructura y significado del lenguaje, lo que permitiría su codificación para que una máquina creada por la IA pueda expresarse y darse a entender con los humanos principalmente (Bertona Noviembre 2005).

La **ingeniería de cómputo** ofreció los dispositivos que permiten hacer realidad las aplicaciones de la inteligencia artificial. Los programas de inteligencia artificial por general son extensos y no funcionarían sin los grandes avances de velocidad y memoria desarrollados por la industria de cómputo. Podemos decir que la ciencia de la computación permite que la inteligencia artificial sea una realidad (Brykczynskki 1999).

3.2 Técnicas de Inteligencia Artificial

El cerebro es el órgano en el que se alojan las habilidades cognoscitivas de los seres humanos. Puede decirse que es un sistema de procesamiento de información extremadamente complejo, cuyo modo de funcionamiento es eminentemente paralelo y cuyo comportamiento no puede describirse por medio de sencillos modelos lineales. Las técnicas de la IA son teorías relativamente nuevas que han generado soluciones muy confiables a problemas de ingeniería altamente complejos. Muchos de estos pueden ser solucionados por métodos tradicionales, pero encuentran en las redes neuronales, la lógica difusa, los sistemas expertos o los algoritmos genéticos una alternativa fácil de implementar y altamente segura. Cada una de estas técnicas pretende explicar a su manera la forma como funciona nuestro cerebro al momento de recibir, procesar y almacenar la información proporcionada por el medio. Un sistema de ordenadores que trabaje con técnicas de la lA deberá estar en capacidad de combinar información de forma **“inteligente”**, alcanzar conclusiones y justificarlas (al igual que el resultado final).

3.2.1 Lógica Difusa

Cuando escuchamos expresiones tales como *la sopa está muy caliente*, *la temperatura es baja,* o *no se es tan viejo*, estamos refiriéndonos a conceptos que para nosotros son entendibles aunque sea información imprecisa, pues el concepto, *temperatura baja,* puede tener valores diferentes para cada individuo, es decir, para un esquimal una temperatura baja puede ser de 4 grados centígrados,

mientras que para un árabe, puede ser de 14 grados centígrados. Igual sucede, con el concepto de edad madura ya que la madurez podría considerarse a los 40 años, pero también podrían entrar a este grupo los de 35 y los de 45, inclusive hasta los de 50 años. Conceptos como éstos, para un ser humano implican una serie de razonamientos que involucran datos e imágenes que le permiten entender el mensaje, pero para una máquina sería imposible procesar información tan inexacta ya que como se acaba de explicar, hay conceptos los cuales pueden tener muchos valores de veracidad o falsedad.

La lógica difusa es una técnica de la IA, que pretende producir resultados exactos a partir de datos imprecisos; el adjetivo *difuso* aplicado a ella se debe a que los valores de verdad no deterministas utilizados tienen una connotación de incertidumbre. Cuando planeamos una reunión de negocios en la tarde, al escuchar el término *en la tarde*, nuestra mente razona y asocia unos datos que le permiten entender el concepto, cosa que a una máquina le sería imposible realizar, porque es incapaz de razonar y comprender un aspecto abstracto o impreciso. Para conseguir que una máquina comprenda estos conceptos, la lógica difusa utiliza una función de pertenencia $\mu A(x)$ que indica el grado en que la variable x está incluida en el concepto representado por la etiqueta A. Para nuestro caso la variable x serían las horas y el conjunto o etiqueta A sería *la tarde*. Hay que tener en cuenta que la función de pertenencia puede adquirir valores entre cero y uno [0,1]. Partimos de la base de que la hora perfecta para decir *en la tarde* se presenta a las 3:00 p.m., en ese caso la función de pertenencia($\mu A(x)$) entre las horas(x) y la tarde(A), será máxima y valdrá 1($\mu A(3)=1$). Sin embargo no podemos descartar las horas de 2:00 p.m. o las 4:00 p.m. como tarde. Por otro lado, las horas antes de las 2:00pm y después de las 4:00pm tampoco se pueden considerar radicalmente que no pertenecen al conjunto; aunque el grado de pertenencia a la misma será mucho menor y cada vez más cercano a cero(por ejemplo $\mu A(4)= 0.7$). Con esta teoría conseguimos que esa relación matemática que hemos obtenido, gracias a la función de

pertenencia, forme la base para que una máquina sea capaz de interpretar, si un elemento pertenece o no a un conjunto difuso y lo que es más importante, que pueda evaluar si ese grado de pertenencia es elevado (cercano a 1) o en cambio es despreciable (cercano a 0). Con la Ilustración 2, pretendemos dar una explicación sencilla del ejemplo mencionado anteriormente.

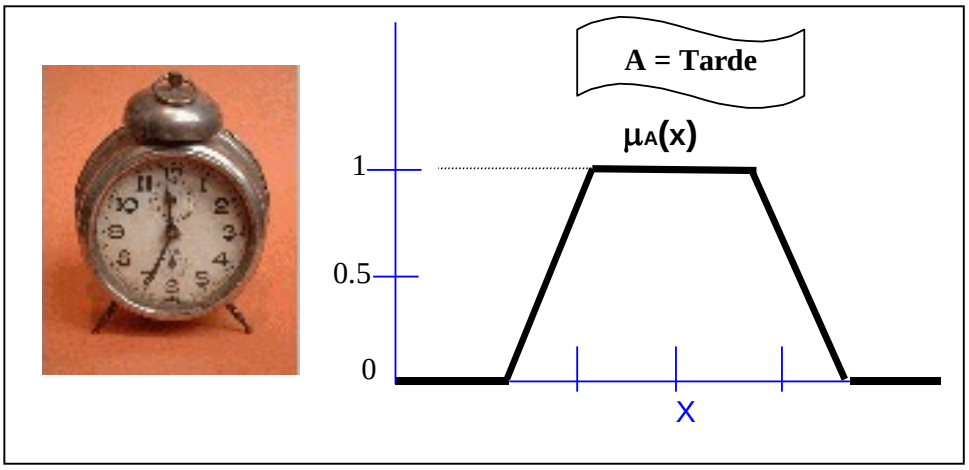


Ilustración 2 Valores de pertenencia del conjunto A (Arias y Cardona 1997)

Ya sabemos que la función de pertenencia μA(x), indica el grado en que la variable x está incluida en el concepto representado por la etiqueta A, pero debemos tener en cuenta que un elemento x puede pertenecer a varios conjuntos o etiquetas, por ejemplo, para la etiqueta A, que involucra las horas de la tarde la variable x = 6 p.m., tiene una función de pertenencia de 0.3, pero para la etiqueta B que sería por ejemplo las horas de la noche la variable x = 6 p.m., tendría un valor de 0.5, con esto pretendemos decir que, en la lógica difusa un elemento puede pertenecer a muchos conjuntos con niveles de pertenencia diferentes para cada etiqueta. Una generalización del ejemplo anterior puede verse representada en la Ilustración 3.

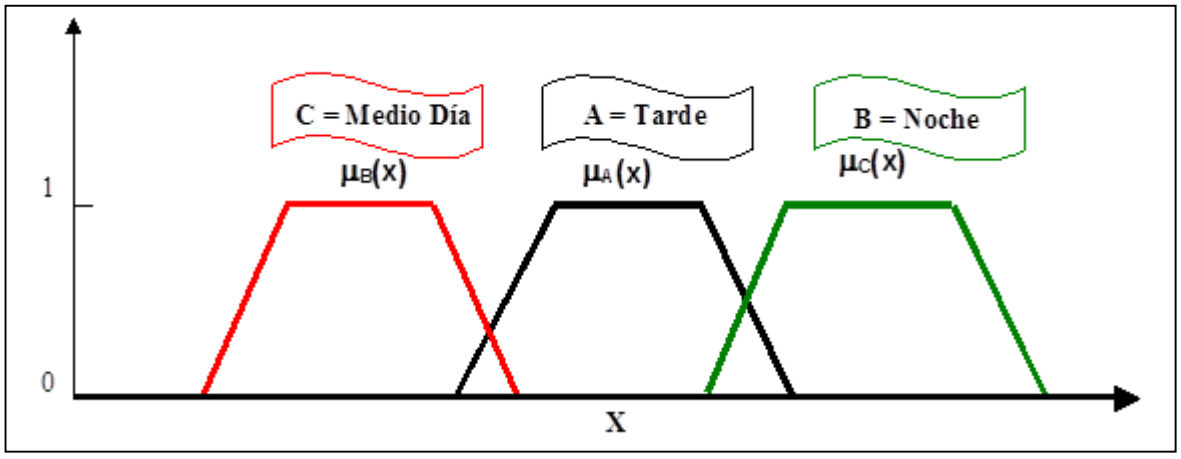


Ilustración 3 Valores de pertenencia de los conjuntos A, B y C (Arias y Cardona 1997).

La Lógica Difusa es considerada una extensión de la Lógica multivaluada, ya que asocia a sus enunciados valores de verdad, que son grados de veracidad o falsedad, mucho más amplios que los valores de verdadero y falso utilizados por la lógica clásica. Estudios anteriores afirman que la fiabilidad de la teoría difusa es muy alta, pero no olvidemos que su objetivo es el de simular el razonamiento humano, así es que, los niveles de éxito en principio son muy bajos, debido a que un computador necesita tener predefinidos todos y cada uno de los posibles inconvenientes que pueden surgir durante el proceso de razonamiento (Arias y Cardona 1997).

En la actualidad encontramos múltiples aplicaciones de la teoría difusa, entre las cuales podríamos mencionar (Arias y Cardona 1997):

- Un metro controlado mediante lógica difusa en Sendai (Japón), el cual tenía la peculiaridad de poseer controladores que hacían la frenada y la aceleración mucho más suaves, facilitando así la conducción.
- Las lavadoras difusas tienen más de 400 ciclos preprogramados; a pesar de su complejidad tecnológica son más fáciles de operar que las lavadoras tradicionales, porque el usuario solo pone en marcha la lavadora, el resto queda en manos del control difuso, éste evalúa automáticamente el material, el

volumen, la suciedad de la ropa, elige el ciclo óptimo de lavado, así como el caudal de agua que ha de emplear.

- Para algunas actividades domésticas cotidianas se inventó un sistema de ventilación que usa el control difuso para conmutar un ventilador según los conocimientos de cantidad de polvo, olores, temperatura y humedad ambiente.

- Los baños difusos, tienen un controlador que mantiene el agua a la temperatura ideal del usuario, ni muy fría ni muy caliente.

3.2.2 Sistema Experto

Los *sistemas expertos* permiten la creación de máquinas capaces de resolver problemas que para su resolución implica un procedimiento basado en el conocimiento; esto comprende las capacidades de utilización de normas o estructuras que contengan conocimientos y experiencias de expertos especializados, deducción lógica de conclusiones e interpretación de datos ambiguos. Son programas de ordenador diseñados para actuar como un especialista humano en un dominio particular o área de conocimiento. En este sentido, pueden considerarse como intermediarios entre el experto humano, que transmite su conocimiento al sistema, y el usuario que lo utiliza para resolver un problema con la eficacia del especialista. El sistema experto utilizará para ello el conocimiento que tenga almacenado y algunos métodos de inferencia. A la vez, el usuario puede aprender observando el comportamiento del sistema. Es decir, los sistemas expertos se pueden considerar simultáneamente como un medio de ejecución y transmisión del conocimiento.

La característica fundamental de un Sistema Experto es que separa la Base de conocimiento (conocimientos almacenados) del motor de inferencia (programa que los controla). También consta una base de datos aparte llamada base de hechos que contiene datos propios de un determinado problema. Otra característica importante es que el sistema es capaz de justificar su propia línea de

razonamiento de forma inteligible para el usuario. También es muy eficaz cuando tiene que analizar una gran cantidad de información, interpretándola y proporcionando una recomendación a partir de la misma (Del Brío 2002).

3.2.3 Agentes Inteligentes

*"Un agente inteligente es un sistema con la capacidad de prever sobre su ambiente una **acción** y dar respuesta o actuar en tal ambiente mediante **reacciones** que aprende o desarrolla con experiencias previas para cumplir con sus objetivos a través del tiempo"*.*

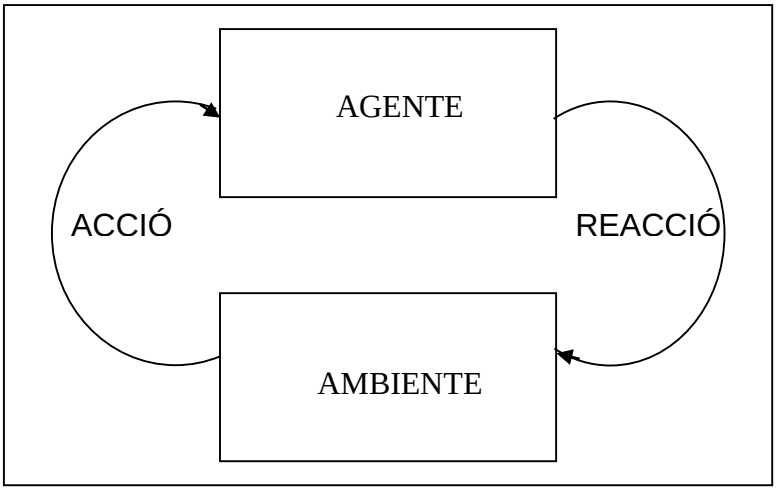


Ilustración 4 Un agente que interactúa con su ambiente

Los agentes humanos a la edad de un año perciben de forma moderadamente clara el ambiente en que crecen, y aprenden de sus errores, es decir, por ejemplo (ver Ilustración 4), un bebe al dar sus primeros pasos por lo general si no tiene alguien quien lo guíe, caerá casi siempre porque encuentra una distracción, perdiendo la concentración de lo que hace, en consecuencia sentirá fuertemente lastimado su mentón, al transcurrir el tiempo y seguir intentándolo, por algún motivo pierden su concentración y va otra vez al suelo, en esta ocasión con un recuerdo o una experiencia anterior no muy agradable, es por esto que de alguna

*Definición desarrollada por la autora.

manera reacciona colocando sus manos de manera firme antes de tocar el suelo para no ver nuevamente afectado su mentón; es en este ejemplo donde observamos lo que podríamos llamar los “primeros pasos” de un agente, cuando este agente haya desarrollado bien su capacidad de prever sobre su ambiente no tendrá, al momento de tropezar por alguna distracción, la necesidad de recurrir a sus manos, ahora lo hará de una forma más rápida y segura, impedirá caerse con sus pies (Del Brío 2002).

De forma muy similar se espera que actúen los agentes programados, aunque su aprendizaje en un principio se vea necesariamente ligado a la interacción con su usuario, es en esta interacción donde pueden surgir agentes más inteligentes o mejor adaptados que otros, con la aptitud para comunicarse y hasta de formar comunidades de agentes.

Es claro qué para que exista comunicación entre agentes, estos deben hablar el mismo lenguaje, para esto se desarrollan tecnologías basadas en estándares para el procesamiento y presentación de la información, como KIF (Knowledge Interchange Format), formato de intercambio de información, que permitan que agentes programados por diferentes autores se entiendan a través de un lenguaje común.

Finalmente, como lo hemos venido recalcando, una capacidad dominante de los buenos agentes es que utilizan su experiencia para ayudar a su usuario. Por ejemplo, no lo enviarían a los hoteles que a otros clientes les han parecido malos. Semejantemente, si usted viaja mucho su agente debe aprender sobre sus propias preferencias: por cuales líneas aéreas usted tiene más preferencia, qué clase de automóvil usted prefiere alquilar, todas esas pequeñas cosas que puedan hacer la diferencia entre un buen y un mal viaje. Pero se debe recordar, los mejores agentes aprenden tanto como usted quisiera que lo hagan (Nascimento 1994).

3.2.4 Algoritmos Genéticos

La evolución, tal y como la conocemos, es básicamente un método de búsqueda entre un número enorme de posibles “soluciones”. En biología las posibilidades están formadas por un conjunto de secuencias genéticas posibles, y las soluciones deseadas, por organismos capaces de sobrevivir y reproducirse en sus entornos. Estas son las razones por las cuales los mecanismos evolutivos son una fuente de inspiración para los algoritmos de búsqueda (Arias y Cardona 1997).

El buen funcionamiento de un organismo biológico depende de muchos criterios, que además varían a medida que el organismo evoluciona, de modo que la evolución está “buscando” continuamente entre un conjunto cambiante de posibilidades. Por ello, podemos considerarla como un método de búsqueda masivamente paralelo, ya que evalúa y cambia millones de especies a la vez. Las reglas de la evolución, aunque de alto nivel, son simples: las especies evolucionan mediante variaciones aleatorias (mutaciones, recombinaciones, etc.) seguidas por la selección natural, donde el mejor tiende a sobrevivir y a reproducirse, propagando así su material genético a posteriores generaciones.

Los algoritmos genéticos son métodos heurísticos de búsqueda inspirados en lo que sabemos acerca del proceso de evolución natural. Si la naturaleza ha sido capaz de generar organismos óptimos para desempeñarse en medios ambientes sumamente complejos, por qué no copiar sus métodos para resolver nuestros propios problemas tratando de encontrarles soluciones que, de alguna manera, sean óptimas. La gran popularidad que han alcanzado los algoritmos genéticos, se debe, en buena parte, a que hacen posible abordar problemas en los que es muy difícil aplicar procedimientos matemáticos tradicionales. La gran plasticidad, inherente a estos métodos, hace posible aplicarlos a la solución de muy diversos problemas si hacer modificaciones sustanciales al procedimiento general, es decir, no asumen nada o casi nada acerca del problema a resolver, a diferencia de los

métodos tradicionales en los que se exige que el modelo matemático del problema consista de una función claramente definida y con ciertas propiedades (Arias y Cardona 1997).

"Algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana" [2].

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución a un problema dado. A cada individuo se le asigna un valor ó puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos, descendientes de los anteriores, los cuales comparten algunas de las características de sus padres (Abdelmalik Moujahid s.f.).

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así, a lo largo de las generaciones, las buenas características se propagan a través de la población, favoreciendo el cruce de los individuos mejor adaptados. Si el Algoritmo Genético ha sido bien diseñado, la, población convergerá hacia una solución óptima del problema (Abdelmalik Moujahid s.f.).

[2] GOLDBERG, D. Genetics Algorithms in Search, Optimization and Machine Learning. Addison Wesley, citado por TOLMOS, Op. cit., p.3.

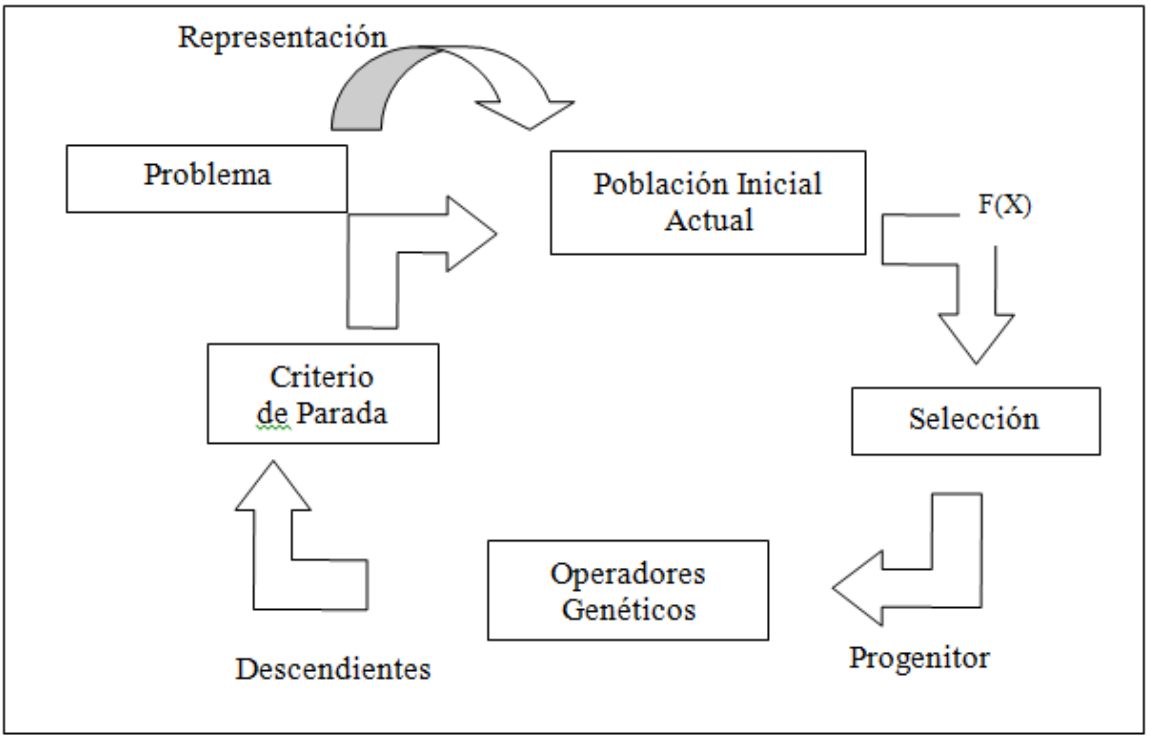


Ilustración 5 Ciclo de un algoritmo genético (Arias y Cardona 1997)

3.3 REDES NEURONALES

Las redes neuronales artificiales (RNA) son una simulación de un sistema nervioso biológico, las cuales pretenden modelar la forma en que el cerebro procesa la información. El cerebro humano es un sistema altamente complejo, no lineal y paralelo; esto significa que puede procesar, organizar y almacenar información simultáneamente; algo que no sucede con los computadores comunes cuyos procesos se ejecutan de un modo secuencial, es decir, un proceso a la vez. Las RNA adquieren el conocimiento a través de la experiencia, el cual es almacenado, del mismo modo que en el cerebro humano; además poseen altísima plasticidad y adaptabilidad, ya que son capaces de cambiar dinámicamente junto con el medio y poseen un alto nivel de tolerancia a fallas, es decir, pueden tener un daño considerable y seguir con un nivel de funcionamiento aceptable. (Arias y Cardona 1997)

Si pensamos en la asombrosa capacidad que tiene el cerebro para reconocer en un plano tridimensional en cuestión de milisegundos a una persona entre muchas otras y cuyo aspecto pudo haber cambiado; o también en la capacidad que éste tiene para responder de forma correcta frente a un estímulo nunca antes recibido o la habilidad de diferenciar a una persona de un objeto. Son éstas y muchas otras características, las que hacen que las redes neuronales se hayan convertido en una gran ayuda en el procesamiento de datos experimentales de comportamiento complejo e iterativo no lineal. (Arias y Cardona 1997)

Los modelos matemáticos en que han sido desarrollados los algoritmos para los tipos de redes son modelos sencillos, que aunque exigen cierto grado de conocimiento de cálculo diferencial, pueden ser asimilados y desarrollados en cualquier lenguaje de programación. Así mismo, las estructuras de las redes se han definido por medio de notación sencilla y comprensible, cada nuevo desarrollo permite cierta flexibilidad en cuanto a la forma final de la red, garantizando su fácil adaptación a aplicaciones particulares (Bigus 1996).

Quizás una de las aplicaciones más populares, donde se han aplicado las RNA, sea en reconocimiento de patrones. Un *patrón,* es una entidad a la que se le puede dar un nombre y que está representada por un conjunto de propiedades medidas y las relaciones entre ellas; por ejemplo, una imagen de una cara humana de las cuales se extrae el vector de características formado por un conjunto de valores numéricos calculados a partir de la misma. El reconocimiento automático, descripción, clasificación y agrupamiento de patrones son actividades importantes en una gran variedad de disciplinas científicas, como biología, sicología, medicina, inteligencia artificial, teledetección, etc.

Dentro de este campo se hace distinción entre patrones estáticos y dinámicos, esto es, aquellos que no contemplan explícitamente la variable tiempo y los que sí respectivamente. En cualquier caso, lo que se busca es un sistema capaz de asignar lo más correctamente posible cada entrada a una clase o patrón. (Arias y Cardona 1997)

En la mayoría de los sistemas de RNA, los esquemas de representación son desarrollados por los diseñadores usando su conocimiento y experiencia en el dominio del problema. Una vez que el sistema de reconocimiento está desarrollado, estos esquemas son inamovibles. Sin embargo, las redes neuronales tienen la propiedad de construir una representación interna de los patrones (*extracción de características*), aunque difícilmente visible. Por esta razón, algunos investigadores alimentan a la red con los datos en bruto (o con un preproceso mínimo, como normalización) y esperan que la propia red extraiga (aprenda) una representación a partir de ellos.

En cualquier caso, una representación adecuada de los datos facilita el proceso de toma de decisión y mejora las tasas de generalización. Sin embargo, el diseño de una buena representación exige un conocimiento profundo de la naturaleza del problema, lo cual no siempre es posible. La forma de aprender un esquema de representación partiendo de un conjunto de datos es todavía un problema abierto.

3.3.1 Modelo Biológico de una Neurona

El cerebro es el elemento principal del sistema nervioso humano y está compuesto por un tipo especial de célula llamada neurona. Una neurona es una célula viva y como tal posee todos los elementos comunes de las células biológicas. A su vez, las neuronas tienen características propias que le permiten comunicarse entre ellas, lo que las diferencia del resto de las células biológicas (Bertona Noviembre 2005).

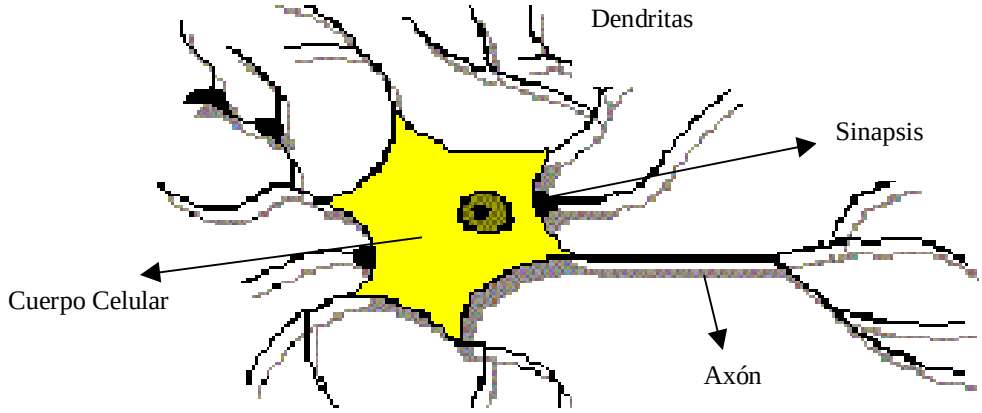


Ilustración 6 Partes de una neurona (Gutiérrez s.f.)

De la ilustración 6 se observa que la neurona biológica está compuesta por un cuerpo celular o soma, del cual se desprende árbol de ramificaciones llamado árbol dendrítico, compuesto por las dendritas. Del soma también parte una fibra tubular, llamada axón, el cual suele ramificarse cerca de su extremo. Las dendritas actúan como un canal de entrada de señales provenientes desde el exterior hacia la neurona, mientras que el axón actúa como un canal de salida (Gutiérrez s.f.).

El espacio entre dos neuronas vecinas se denomina sinapsis. En el córtex cerebral se observa una organización horizontal en capas, así como también una organización vertical en columnas de neuronas.

La intensidad de una sinapsis no es fija, sino que puede ser modificada en base a la información proveniente del medio. De esta manera la estructura del cerebro no permanece fija sino que se va modificando por la formación de nuevas conexiones, ya sean excitadoras o inhibidoras, la destrucción de conexiones, la modificación de la intensidad de la sinapsis, o incluso por muerte neuronal. Desde un punto de vista funcional, las neuronas conforman un procesador de información sencillo. Constan de un subsistema de entrada (dendritas), un subsistema de procesamiento (el soma) y un subsistema de salida (axón) (Bertona Noviembre 2005).

La comunicación entre neuronas se desarrolla de la siguiente manera: en el soma de las neuronas transmisoras o presinápticas se genera un pulso eléctrico llamado

potencial de acción. El pulso eléctrico se propaga a través del axón en dirección a las sinapsis. La información se transmite a las neuronas vecinas utilizando un proceso químico, mediante la liberación de neurotransmisores. Estos neurotransmisores se transmiten a través de la sinapsis hacia la neurona receptora. La neurona receptora o postsináptica toma la señal enviada por cientos de neuronas a través de las dendritas y la transmite al cuerpo celular. Estas señales pueden ser excitadoras (positivas) o inhibidoras (negativas) (Gurney 1997). El soma es el encargado de integrar la información proveniente de las distintas neuronas. Si la señal resultante supera un determinado umbral (umbral de disparo) el soma emite un pulso que se transmite a lo largo del axón dando lugar a la transmisión eléctrica a lo largo de la neurona. Al llegar la señal al extremo del axón se liberan neurotransmisores que permiten transmitir la señal a las neuronas vecinas. (Nascimento 1994)

3.3.2 Sistema Neuronal Artificial

Las redes neuronales son modelos matemáticos que intentan reproducir el funcionamiento del cerebro humano. El objetivo principal de un Sistema Neuronal Artificial es la construcción de sistemas capaces de presentar un comportamiento inteligente es decir, la capacidad de aprender a realizar una determinada tarea.

Las características principales que reproducen las redes neuronales artificiales se pueden reducir a los siguientes tres conceptos: procesamiento paralelo, distribuido y adaptativo. (Del Brío 2002)

La eficacia de este modelo radica en el procesamiento paralelo realizado por las neuronas artificiales. La neurona artificial es un elemento de procesamiento simple y constituye el elemento principal de un sistema neuronal artificial.

Las neuronas artificiales se organizan en estructuras denominadas capas. Una red neuronal artificial esta un compuesta por un conjunto de capas. De este modo, la información se encuentra distribuida a lo largo de las sinapsis de la red, dándole a este sistema cierta tolerancia a fallos. Las redes neuronales artificiales son

capaces de adaptar su funcionamiento a distintos entornos modificando sus conexiones entre neuronas. De esta manera pueden aprender de la experiencia y generalizar conceptos. Por último, un grupo de redes neuronales, junto con las interfaces de entrada y salida, y los módulos lógicos adicionales conforman un sistema neuronal artificial.

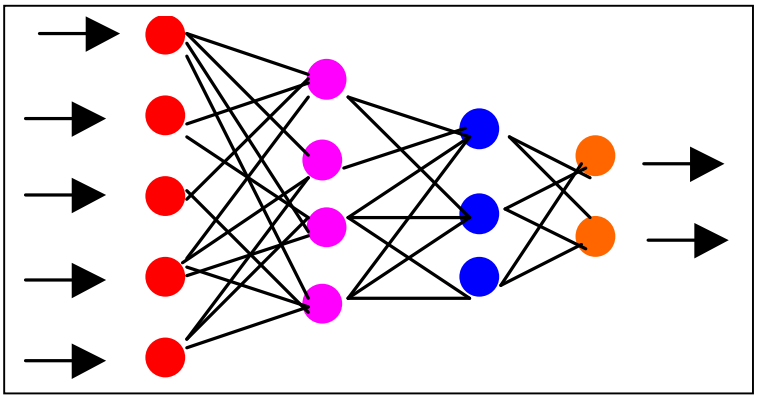


Ilustración 7 Red neuronal artificial

3.3.3 Modelo de una Neurona Artificial

La neurona artificial tiene un procesamiento simple que a partir de un vector de entradas produce una única salida. Podemos encontrar tres tipos de neuronas artificiales, donde cada una de las cuales tiene su contraparte en el sistema nervioso (Arias y Cardona 1997):

Neuronas de Entrada: reciben información directamente desde el exterior.

Neuronas Ocultas: las que reciben información desde otras neuronas artificiales. Es en estas neuronas, específicamente en sus sinapsis, donde se realiza la representación de la información almacenada.

Neuronas de Salida: las que reciben la información procesada y las devuelven al exterior.

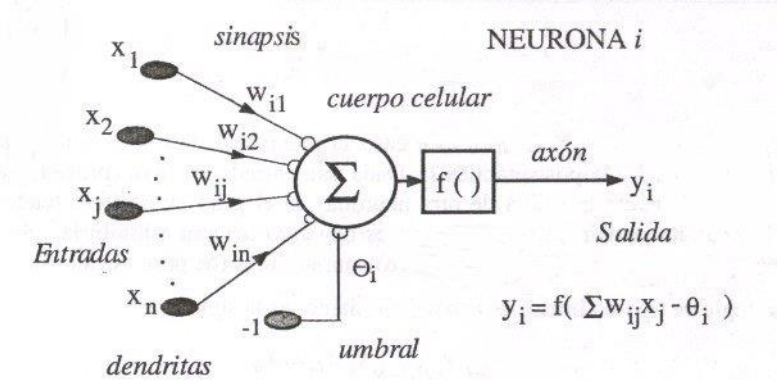


Ilustración 8 Neurona Artificial tipo McCulloch-Pitts

3.3.4 Componentes de una Neurona Artificial

Conjunto de entradas, $x_j(t)$. Estas pueden ser provenientes del exterior o de otras neuronas artificiales. Las entradas y salidas de una neurona pueden ser clasificadas en dos grandes grupos, binarias o continuas. Las neuronas binarias (digitales) sólo admiten dos valores posibles. En general en este tipo de neurona se utilizan los siguientes dos alfabetos {0,1} o {-1,1}. Por su parte, las neuronas continuas (analógicas) admiten valores dentro de un determinado rango, que en general suele definirse como [-1, 1]. La selección del tipo de neurona a utilizar depende de la aplicación y del modelo a construir.

Peso sinápticos, w_{ij}. Define la fuerza de una conexión sináptica entre dos neuronas, la neurona presináptica i y la neurona postsináptica j. Los pesos sinápticos pueden tomar valores positivos, negativos o cero. En caso de una entrada positiva, un peso positivo actúa como excitador, mientras que un peso negativo actúa como inhibidor. En caso de que el peso sea cero, no existe comunicación entre el par de neuronas.
Mediante el ajuste de los pesos sinápticos la red es capaz de adaptarse a Cualquier entorno y realizar una determinada tarea.

Regla de propagación, $\sigma_i(w_{ij}, x_j(t))$. Integra la información proveniente de las distintas neuronas artificiales y proporciona el valor del potencial postsináptico de

la neurona i. La regla de propagación determina el potencial resultante de la interacción de la neurona i con las N neuronas vecinas.

Ecuación 1 Potencial resultante h_i

$$h_i(t) = \sigma_i(w_{ij}, x_j(t))$$

Ecuación 2 Regla de propagación más simple y utilizada

$$h_i(t) = \sum_j w_{ij} * x_j(t)$$

Función de activación, $f_i(a_i(t-1), h_i(t))$. Determina el estado de activación actual de la neurona en base al potencial resultante **h_i** y al estado de activación anterior de la neurona **$a_i(t-1)$**. El estado de activación de la neurona para un determinado instante de tiempo t puede ser expresado de la siguiente manera:

Ecuación 3 Función de Activación

$$a_i(t) = f_i(a_i(t-1), h_i(t))$$

Sin embargo, en la mayoría de los modelos se suele ignorar el estado anterior de la neurona, definiéndose el estado de activación en función del potencial resultante **hi**:

Ecuación 4 Estado de activación en función del potencial resultante hi

$$a_i(t) = f_i(h_i(t))$$

Tabla 1 Funciones de Activación

Función	Formula	Rango
Identidad	$x = y$	$[-\infty, \infty]$
Escalón	$y = \begin{Bmatrix} +1 \; si \; x \geq 0 \\ 0 \; si \; x < 0 \end{Bmatrix}$ $y = \begin{Bmatrix} +1 \; si \; x \geq 0 \\ -1 \; si \; x < 0 \end{Bmatrix}$	$[-1, 1]$
Lineal a tramos	$y = \begin{Bmatrix} x & si -1 \leq x \leq 1 \\ +1 & si \; x > 1 \\ -1 & si \; x < -1 \end{Bmatrix}$	$[-1, 1]$
Sigmoidea	$y = \frac{1}{1 + e^{-x}}$ $y = \tanh(x)$	$[0, 1]$ $[-1, 1]$
Sinusoidal	$y = \sin(\omega x + \varphi)$	$[-1, 1]$

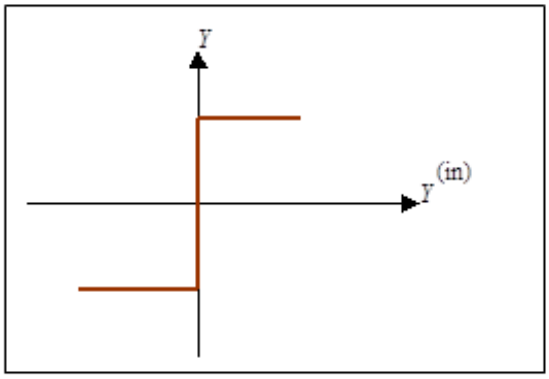


Ilustración 9 Función de tipo escalón

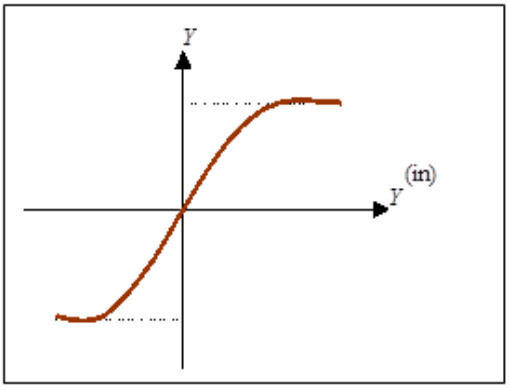


Ilustración 10 Función de tipo Sigmoidea

Las Ilustraciones 13 y 14 son funciones de activaciones típicas, no lineales. Las neuronas y sus funciones de activación se dividen en dos tipos: bipolares y binarias. A veces se suele usar como función de activación una relación lineal, generalmente una función de identidad. Esta se usa por lo general para neuronas de entrada a la red o sensores. Generalmente si la función de activación de una neurona es lineal, decimos que es una neurona lineal y la representamos con la

forma de un cuadrado. En caso contrario decimos que la neurona es no lineal y la representamos con un círculo.

Función de salida. La función de salida proporciona el valor de salida de la neurona, en base al estado de activación de la neurona.

Ecuación 5 Función de Salida de una Neurona

$$y_i(t) = F_i(a_i(t)) = a_i(t)$$

3.3.5 Fases de Operación de una Red Neuronal

Durante la operatoria de una red neuronal podemos distinguir claramente dos fases o modos de operación: la fase de aprendizaje o entrenamiento, y la fase de operación o ejecución.
Durante la primera fase, la fase de aprendizaje, la red es entrenada para realizar un determinado tipo de procesamiento. Una vez alcanzado un nivel de entrenamiento adecuado, se pasa a la fase de operación, donde la red es utilizada para llevar a cabo la tarea para la cual fue entrenada (Bertona Noviembre 2005).

Fase de entrenamiento: Una vez seleccionada el tipo de neurona artificial que se utilizará en una red neuronal y determinada su topología es necesario entrenarla para que la red pueda ser utilizada. Partiendo de un conjunto de pesos sinápticos aleatorio, el proceso de aprendizaje busca un conjunto de pesos que permitan a la red desarrollar correctamente una determinada tarea. Durante el proceso de aprendizaje se va refinando iterativamente la solución hasta alcanzar un nivel de operación suficientemente bueno.

El proceso de aprendizaje se puede dividir en tres grandes grupos de acuerdo a sus características (Isasi Viñuela 2004)**,** (X 1999):

- Aprendizaje supervisado: Se presenta a la red un conjunto de patrones de entrada junto con la salida esperada. Los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada.

- Aprendizaje no supervisado. Se presenta a la red un conjunto de patrones de entrada. No hay información disponible sobre la salida esperada. El proceso de entrenamiento en este caso deberá ajustar sus pesos en base a la correlación existente entre los datos de entrada.

- Aprendizaje por refuerzo. Este tipo de aprendizaje se ubica entre medio de los dos anteriores. Se le presenta a la red un conjunto de patrones de entrada y se le indica a la red si la salida obtenida es o no correcta. Sin embargo, no se le proporciona el valor de la salida esperada. Este tipo de aprendizaje es muy útil en aquellos casos en que se desconoce cuál es la salida exacta que debe proporcionar la red (Bertona Noviembre 2005).

Fase de operación.

Una vez finalizada la fase de aprendizaje, la red puede ser utilizada para realizar la tarea para la que fue entrenada. Una de las principales ventajas que posee este modelo es que la red aprende la relación existente entre los datos, adquiriendo la capacidad de generalizar conceptos. De esta manera, una red neuronal puede tratar con información que no le fue presentada durante de la fase de entrenamiento (Bertona Noviembre 2005).

3.3.6 Tamaño de una red neuronal (L 1992).

Cuando trabajamos con grandes cantidades de neuronas, es natural ordenar aquella que tienen comportamientos similares en “capas”. De ahí que se utilicen los subíndices para las neuronas, donde cada capa es un vector de neuronas.

- **Redes Unicapa.** Se acostumbra no contabilizar la capa de entrada, por lo tanto se dice que la red es unicapa. Con una red unicapa solo se pueden resolver problemas para el reconocimiento de patrones pero que sean linealmente separables. En redes de este tipo las neuronas pueden ser lineales o no lineales. La Ilustración 17 muestra el modelo de una red unicapa.
- **Red Multicapa.** En una Red Multicapa, las capas ocultas corresponde a la capa 2, siempre son no lineales. Se puede demostrar que si se construye una red multicapa con capas ocultas lineales, esta es equivalente a una unicapa. La Ilustración 17 muestra el modelo de una red multicapa.

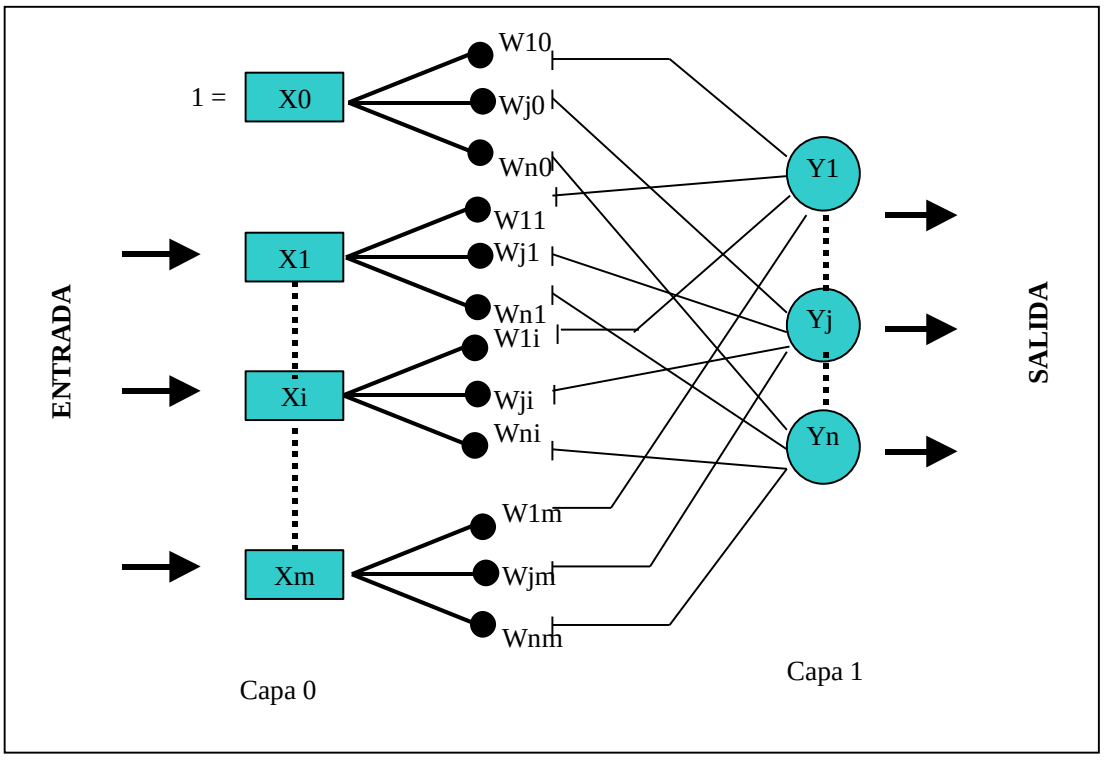


Ilustración 17 Modelo de una red unicapa (L 1992)

Cada neurona de una capa no necesita de las demás en su misma capa para trabajar, son capaces por lo tanto de trabajar simultáneamente. Esto permite que en la red neuronal las neuronas trabajen paralelamente. Una red multicapa es

capaz de resolver problemas más complejos, pero su proceso de aprendizaje es más complicado.

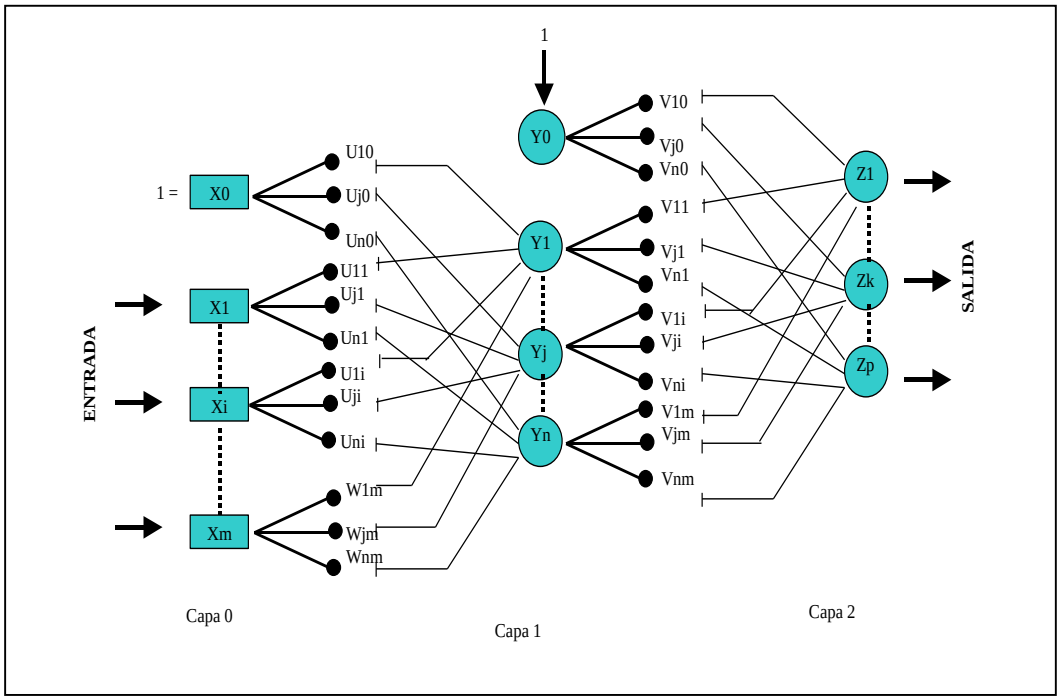


Ilustración 11 Modelo de una red multicapa (L 1992)

3.3.7 Algunas aplicaciones de las RNA

- **Redes neuronales en reconocimiento de locutor.** El reconocimiento automático del locutor (RAL) o reconocimiento automático del hablante es una área de investigación y desarrollo de aplicaciones de gran importancia actualmente. El RAL persigue el reconocimiento de las personas mediante su voz, sin que sea necesaria la intervención de un operador humano. Como es evidente, el ordenador será el encargado de sustituir al operador humano en la tarea de reconocimiento. Es fácil ver, que el RAL se enmarca dentro de un área de trabajo más amplia: el reconocimiento de personas mediante parámetros biológicos cuyo valor es diferente en cada persona, como por ejemplo el iris, la forma de la cara o las huellas dactilares.

- **Redes neuronales en percepción remota.** La percepción remota comprende un conjunto de conocimientos y técnicas utilizadas para determinar características físicas, químicas y biológicas de los objetos a través de mediciones realizadas a distancia, sin un contacto material con ellos. Ha probado ser, a nivel mundial, una herramienta poderosa en la comprensión global de fenómenos, la cual es particularmente apreciada por sus características de técnica no invasiva y no destructiva, así como de cobertura global. Dentro de éste contexto, la radiometría satelital se ha convertido en una herramienta útil en tareas como caracterización, prospección y monitoreo continuo de recursos naturales, dado que permiten una perspectiva global desde los puntos de vista espacial y temporal.

- **Reconocimiento de patrones a partir de imágenes aéreas.** La fotointerpretación es una herramienta útil para reconocer patrones y realizar diversos tipos de Planificación Territorial a partir de fotografías aéreas. La clasificación y cuantificación de estos patrones a partir de las fotografías aéreas tiene importancia en la construcción de mapas temáticos y se viene realizando hace algunas décadas. Pero la década de los 80 y los 90 fueron especialmente exigentes por la creciente necesidad del estudio de la Biodiversidad y con esto la creación de Parques Nacionales o Reservas que posean una diversidad de ecosistemas y hábitats. Paralelamente, el desarrollo de la tecnología computacional abarató los costos y fue posible preguntarse sobre la factibilidad de incorporar a la máquina y sus potencialidades en los procesos de clasificación de la cobertura del terreno, usando como insumos las fotografías aéreas con el propósito de realizar mapas de la cobertura y establecer pautas para la cuantificación de la diversidad de las regiones con el fin de generar información útil para su manejo sostenible.

- **Redes Neuronales para la Toma de Decisiones.** En las implementaciones de la IA convencional, las condiciones y acciones que entran en el problema son descritas como decisión árbol y la evaluación es un problema de combinatoria. Sin embargo, este no es exactamente el camino mediante el que piensa un ser natural. Éste debe hacer análisis formales similares para evitar malas soluciones, pero cuando se llega a la estrategia formal, entonces otras razones, basadas en presentimientos e instintos intuitivos sobre la situación, adquieren más importancia. Estas capacidades pueden, no obstante realizarse en sistemas de aprendizaje artificial suficientemente grandes que operen con principios de computación neuronal. El criterio de operación aplicado en estos casos es más complejo y será aprendido automáticamente de ejemplos de algún tipo de descripción estadística.

  Se puede decir que las estrategias de toma de decisiones están almacenadas en forma de reglas, mientras que estas reglas se establecen automáticamente, y sólo existen en forma implícita como un estado colectivo de las interconexiones adaptativas (Martinez 2000).

- **Predicción de acciones.** Consiste en el desarrollo de una red neuronal capaz de realizar la predicción del precio de las acciones para un número dado de compañías. Esta predicción se realiza mediante redes alimentadas hacia adelante, y el objetivo en este particular caso es predecir el siguiente valor en la serie de tiempo: el próximo precio de la acción (Fleur W. Op 't Landt 1997).

- **Predicción de tráfico vehicular.** Se han utilizado redes neuronales recurrentes para la predicción a corto plazo del tráfico en una carretera, con el fin de prevenir congestiones y tener un control del acceso a la autopista.

Se utilizaron datos estimados de otros días con propiedades similares; y se pudo verificar que las redes neuronales resolvieron este tipo de predicción y obtuvieron mejores resultados que los métodos estadísticos convencionales (GARZON y HERRADA 2006).

- **Predicción del tráfico en una autopista.** Dierdre predice el volumen y el nivel de ocupación de una autopista, los resultados obtenidos fueron mejores que los obtenidos por técnicas usadas anteriormente. Se demostró que las redes neuronales con exactitud pueden predecir el volumen y la ocupación un minuto por adelantado, se utiliza una red neuronal multicapa entrenándose mediante retropropagación, este trabajo se auxilia de una rampa de medición de lógica difusa (*fuzzy logic ramp metering*) para realizar la predicción (Deirdre R. Meldrum 1995).

- **Predicción de la transición mensual del índice de precios de acciones.** Usando recurrencia y retropropagación, la red neuronal es entrenada para aprender conocimiento experimental y para predecir la transición de precios de acciones, tomando como entrada principal algunos indicadores económicos y obteniendo la transición relativa de la composición del índice de precios de acciones, usando ocho años de datos económicos, la predicción del crecimiento mensual o la caída del índice del precio de acciones. Los resultados indican que las redes neuronales son herramientas eficientes para la predicción del precio de las acciones (Park 1992).

- **Redes neuronales para reconocimiento de patrones a mano alzada:** Las redes neuronales que están diseñadas para trabajar con patrones, pueden ser clasificadoras de patrones o asociadoras de patrones. Las redes pueden tomar un vector (una serie de números) y entonces clasificarlo. Un reconocedor de letras toma un vector y envía otro a la

salida, por ejemplo, tomando una imagen 'sucia' (sin ser procesada) y en vía a la salida la imagen que representa la más próxima de las aprendidas, a la imagen de entrada. En un nivel más práctico, las redes neuronales pueden ser usadas en aplicaciones mucho más complejas como el reconocimiento de firmas/caras/huellas digitales[3].

3.3.8 Redes neuronales con conexión hacia delante (*feed-forward*)

Es la forma más básica de red. Las neuronas de cada nivel sólo están conectadas con las neuronas de los niveles posteriores por lo que la información se propaga hacia delante. Algunas de las redes de este tipo más conocidas son: Perceptron, Adaline (ADAptative LINear Elements), Madaline[4] (Multiple Adaline) y el MLP. Todas ellas suelen aplicarse a la aproximación de funciones o al reconocimiento o clasificación de patrones.

3.3.8.1 Perceptron

A finales de 1950 Frank Rosenblatt y otros investigadores desarrollaron una clase de redes neuronales llamadas perceptrones. Las neuronas de estas redes eran similares a las de McCulloch y Pitts. La contribución clave de Rosenblatt fue la introducción de una regla de aprendizaje para la formación de redes perceptron para resolver problemas de reconocimiento de patrones. Demostró que su regla de aprendizaje siempre convergirá a los pesos correctos de la red, si existen pesos que solucionan el problema. El Perceptron pudo incluso aprender cuando se inicializaba con valores aleatorios de sus pesos.

Las limitaciones del Perceptron fueron publicadas en el libro Perceptrons por Marvin Minsky y Seymour Papert. Ellos demostraron que las redes perceptron eran incapaces de implementar ciertas funciones elementales. No fue sino hasta la

[3] http://repositorio.puce.edu.ec/bitstream/handle/22000/1092/T-PUCE-0664.pdf?sequence=1&isAllowed=y

[4] Los modelos Adaline y Madaline (Widrow 1959), permitieron usar, por primera vez, una red neuronal en un problema del mundo real: filtros adaptativos que eliminan ecos en las líneas telefónicas.

década de los 8O's que estas limitaciones fueron superadas con las redes perceptron mejoradas (multicapa) asociadas con reglas de aprendizaje.

Su importancia histórica radica en que fue el primer modelo en poseer un mecanismo de entrenamiento que permite determinar automáticamente los pesos sinápticos que clasifican correctamente a un conjunto de patrones a partir de un conjunto de ejemplos.

La arquitectura del perceptron está compuesta por dos capas de neuronas, una de entrada y una de salida. La capa de entrada es la que recibe la información proveniente del exterior y la transmite a las neuronas sin realizar ningún tipo de operación sobre la señal de entrada. En general la información entrante es binaria. La función de activación de las neuronas de un perceptron es del tipo escalón, dando de esta manera sólo salidas binarias. Cada neurona de salida del perceptron representa a una clase. Una neurona de salida responde con 1 si el vector de entrada pertenece a la clase a la que representa y responde con 0 en caso contrario (Bertona Noviembre 2005). La operación de un perceptron con n neuronas de entrada y m neuronas de salidas puede ser resumida de la siguiente manera:

Ecuación 6 Salida de un perceptron con n neuronas de entrada y m neuronas de salidas

$$y_i(t) = f(\sum_{j=1}^{n} w_{ij} x_j - \theta_i) \quad \forall i,\ 1 \leq i \leq m$$

El algoritmo de entrenamiento del perceptron se encuentra dentro de los denominados algoritmos por corrección de errores. Este tipo de algoritmos ajusta los pesos de manera proporcional a la diferencia entre la salida actual proporcionada por la red y la salida objetivo, con el fin de minimizar el error producido por la red.

Se puede demostrar que este método de entrenamiento converge siempre en un tiempo finito y con independencia de los pesos de partida, siempre que la función a representar sea linealmente separable. El principal problema de este método de entrenamiento es que cuando la función a representar no es linealmente separable

el proceso de entrenamiento oscilará y nunca alcanzará la solución. Las funciones no separables linealmente no pueden ser representadas por un perceptron (Bertona Noviembre 2005).

Regla de Aprendizaje del Perceptron

El algoritmo de aprendizaje del perceptron es de tipo supervisado, lo cual requiere que sus resultados sean evaluados y se realicen las oportunas modificaciones del sistema si fuera necesario. Los valores de los pesos pueden determinar el funcionamiento de la red; estos valores se pueden fijar o adaptar utilizando diferentes algoritmos de entrenamiento de la red. Se pueden utilizar Perceptrones como máquinas de aprendizaje. Desgraciadamente, no puede aprender a realizar todo tipo de clasificaciones ya que usa un separador lineal como célula de decisión, con lo cual no es posible realizar sino una sola separación lineal (por medio de un hiperplano[5]).

Aprendizaje del perceptron por corrección de error

A continuación veremos el algoritmo de convergencia de ajuste de pesos para realizar el aprendizaje de un Perceptron con N elementos procesales de entrada y un único elemento procesal de salida:

1. *Inicialización de los pesos y del umbral: inicialmente se asigna valores aleatorios a cada uno de los pesos w_i y al umbral ($-w_0 = \theta$).*
2. *Presentación de un nuevo par (Entrada, Salida esperada): Presentar un nuevo patrón de entrada $X_p = (x_1, x_2, \ldots, x_N)$ junto con la salida esperada d(t)*
3. *Cálculo de la salida actual*

$$y(t) = f[\sum_i w_i(t)x_i(t) - \theta]$$

[5] En un espacio bidimensional (como el plano *xy*), un hiperplano es una recta: divide el plano en dos mitades.

Siendo $f(t)$ la función de transferencia escalón.

4. *Adaptación de los pesos*

$$w(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$$
$$o \leq i \leq N$$

Donde $d(t)$ representa la salida deseada. Como factor de aprendizaje (o ganancia) α se establece un valor mayor que cero y menor o igual a uno. Cuanto más pequeño sea α, más pequeños serán los incrementos de los pesos, por lo que se llegará a la solución más despacio. Pero también de forma más segura.

5. *Repetición de los pasos anteriores hasta que todos los patrones de entrada produzcan la salida esperada.*

3.3.8.2 ADALINE/MADALINE

La topología de la red ADALINE es similar a la del perceptron sólo que en este caso la función de salida de las neuronas es lineal. Dado que las señales de entrada pueden ser continuas, la red ADALINE es un dispositivo de entrada/salida analógica (continua) a diferencia del perceptron que de acuerdo a lo dicho anteriormente es un dispositivo entrada/salida digital (binaria).

La operación de una red ADALINE con n neuronas de entrada y m neuronas de salidas puede ser resumida de la siguiente manera:

Ecuación 7 Operación de una red ADALINE

$$y_i(t) = \sum_{j=1}^{n} w_{ij} x_j - \theta_i \quad \forall i,\ 1 \leq i \leq m$$

Sin embargo, la principal diferencia entre la red ADALINE y el perceptron consiste en la regla de aprendizaje que utilizan. En el caso de la red ADALINE implementa como método de aprendizaje la regla de Widrow-Hoff, también conocida como regla LMS (Least Mean Squares, mínimos cuadrados), que realiza una

actualización continua de los pesos sinápticos de acuerdo a la contribución de cada neurona sobre el error total de la red.

Este método produce un conjunto de pesos sinápticos óptimos desde el punto de vista de los mínimos cuadrados (un conjunto de pesos que minimiza el error cuadrático que comete la red), y en caso de que los vectores de entrada sean linealmente independientes produce una asociación perfecta entre entradas-salidas.

Existe una versión multicapa de la ADALINE denominada MADALINE (Multiple ADALINE, múltiples Adalides) que consiste en una red neuronal con neuronas similares a las de la ADALINE pero que contiene capas de neuronas ocultas (Bertona Noviembre 2005).

3.3.8.3 Perceptron Multicapa

Minsky y Papert (S. 1969), realizaron un estudio detallado de los tipos de representaciones posibles con el perceptron, y demostraron que, en un gran número de casos, estos tipos de redes son incapaces de resolver los problemas de clasificación. Por otro lado, también demostraron que un perceptron constituido por varias capas, puede realizar cualquier aplicación desde las capas de entrada a las de salida. El inconveniente que se tenía en estos primeros estadios del manejo del perceptron, era el desconocimiento de un algoritmo de entrenamiento eficiente para redes con más de una capa.

El perceptron multicapa[6] (MLP) es una extensión del perceptron simple. Su topología está definida por un conjunto de capas ocultas, una capa de entrada y una de salida. No tienen restricciones sobre la función de activación aunque en general se suelen utilizar funciones sigmoideas.

[6] Siglas inglesas de Multilayer Perceptron (MLP)

Ecuación 8 Operación de un perceptron multicapa con una única capa oculta

$$z_k = \sum_j w'_{kj}\, y_i - \theta'_i = \sum_j w'_{kj}\, f\left(\sum_i w_{ji} x_i - \theta_i\right) - \theta'_i$$

El perceptron multicapa es una red acíclica de propagación directa con una o más capas de nodos entre las entradas y las salidas. Estas capas adicionales contienen nodos denominados *ocultos,* ya que no son visibles directamente ni desde las entradas, ni desde las salidas.

Arquitectura: las neuronas se encuentran organizadas en distintas capas (Ilustración 21):

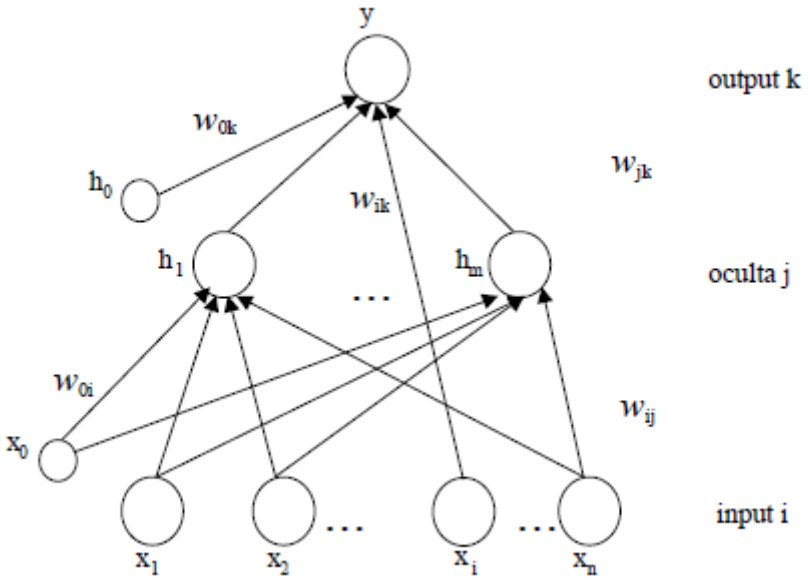


Ilustración 12 Capas de una red neuronal

Capa de entrada: Recibe los inputs del exterior. En esta capa normalmente no se procesan las señales recibidas sino que éstas son enviadas a la capa siguiente. El número de inputs que debe utilizarse depende del problema específico que estemos considerando. Desde el punto de vista de la econometría, las variables independientes serían los inputs del problema que presentaríamos a la capa de entrada.

Capa(s) oculta(s): Las neuronas de esta(s) capa(s) transforman la señal recibida y la envían a la capa de salida. Para la mayoría de los problemas es suficiente con una sola capa oculta. De la modificación de los pesos de las neuronas de esta capa depende el aprendizaje de la red.

Capa de salida: Una vez la red ha transformado la señal, esta capa envía la respuesta al exterior. Las neuronas de esta capa realizan una nueva transformación de la señal recibida por la capa(s) oculta(s). (González 2003)

Existen tres etapas en la aplicación del MLP:

- La *etapa de entrenamiento*, consiste en hacer que la red sea capaz de extraer, a partir de ejemplos reales, reglas generales que la permitan en el futuro responder adecuadamente a patrones nunca vistos anteriormente. Este proceso consiste, por tanto, en la búsqueda de la matriz de pesos óptima.
- La etapa *de validación* o de prueba, es la que ayuda a seleccionar cuál de las redes entrenadas responde mejor a los objetivos planteados.
- *Etapa de evaluación,* una vez elegida la red definitiva que procesará las entradas, ésta resultará operativa, y es entonces cuando la arquitectura, número de neuronas, conexiones y pesos quedan fijos y la red está lista para funcionar. Es en este momento cuando se le presenta a la red *un patrón de entradas nunca visto antes por ella* con el fin de obtener una predicción o clasificación insesgada de los datos. Se evalúa así, una vez comparada la respuesta real y la estimada, la precisión del MLP para la resolución del problema planteado (González 2003).

3.3.9 Redes de propagación hacia atrás (*feedback*)

En estas redes, las neuronas pueden estar conectadas indistintamente con neuronas de niveles previos, posteriores, de su mismo nivel (conexiones laterales) o incluso con ellas mismas (autorrecurrentes). Algunos de los modelos más

conocidos, (L 1992) son: La Bidirectional Associative Memory (BAM) y la máquina de Boltzman.

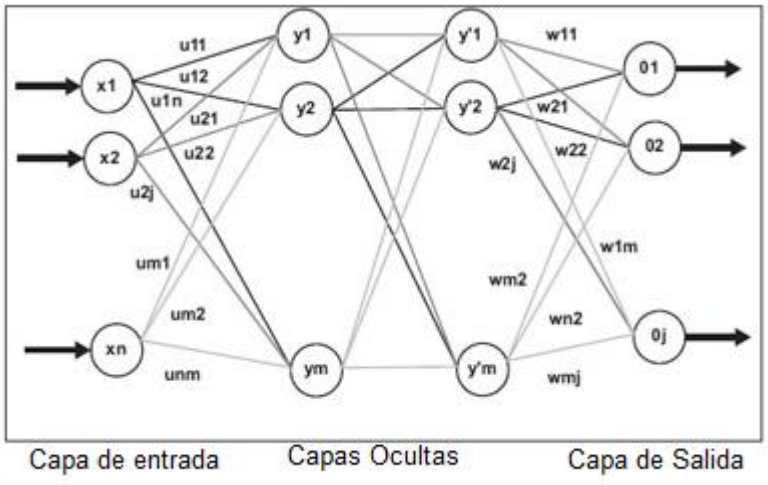


Ilustración 13 Redes de propagación hacia atrás (Martinez 2000)

3.3.10 El Algoritmo *Backpropagation (*Aprendizaje del MLP*)*

En 1986, Rumelhart, Hinton y Williams, formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada y las clases correspondientes, utilizando varios niveles de neuronas.

El método backpropagation (propagación del error hacia atrás), basado en la generalización de la regla delta, a pesar de sus limitaciones, ha ampliado de forma considerable el rango de aplicaciones de las redes neuronales. El funcionamiento de la red backpropagation (BPN)[7] consiste en el aprendizaje de un conjunto predefinido de pares de entradas-salidas dados como ejemplo: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado en las neuronas de salida con la salida que se desea obtener y se calcula un valor de error para cada neurona de salida. A continuación, estos errores se transmiten hacia atrás, partiendo de la capa de salida hacia todas las neuronas de la capa intermedia que contribuyan

[7] Back Propagation Net

directamente a la salida. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aportación relativa al error total.

Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida esté más cercana a la deseada (Martinez 2000).

El algoritmo de retropropagación es un algoritmo basado en el paradigma de aprendizaje supervisado, y se basa en un mecanismo de corrección de errores en el que los errores de la capa de salida se propagan a las capas interiores para realizar la corrección de los pesos de estas capas. El proceso de aprendizaje de una red MLP por backpropagation se divide en dos fases: una fase de la red de alimentación, conocida como fase hacia adelante el que la información se propaga hacia delante, y una fase de corrección de los pesos en la dirección inversa o la espalda propagación de errores , conocida como fase hacia atrás.

3.3.10.1 Regla Delta Generalizada

Fue creada para generalizar la Regla Delta sobre *Redes Neuronales de múltiples capas* y *funciones de transferencia no lineales y diferenciables*.

El aprendizaje es también *Supervisado*. Su utilización consiste en ajustar pesos y bias tratando de minimizar la suma del error cuadrático. Esto se realiza de forma continua, cambiando dichas variables en la dirección contraria a la pendiente del error. Esto se denomina Técnica del Gradiente Descendente.

Redes Neuronales entrenadas mediante esta regla, dan razonables respuestas cuando al sistema se le presentan entradas que no ha visto nunca. Típicamente a una nueva entrada, le hará corresponder una salida similar a la salida obtenida para un patrón de entrenamiento, siendo éste similar al patrón presentado a la red. Una de las grandes propiedades de este tipo de estructuras es su capacidad de *Generalización.*

3.3.10.2 Estructura y Aprendizaje de la Red Backpropagation

En una red Backpropagation existe una capa de entrada con n neuronas y una capa de salida con m neuronas y al menos una capa oculta de neuronas internas. Cada neurona de una capa (excepto las de entrada) recibe entradas de todas las neuronas de la capa anterior y envía su salida a todas las neuronas de la capa posterior (excepto las de salida). No hay conexiones hacia atrás feedback ni laterales entre las neuronas de la misma capa.

La aplicación del algoritmo tiene dos fases, una hacia delante y otra hacia atrás. Durante la primera fase el patrón de entrada es presentado a la red y propagado a través de las capas hasta llegar a la capa de salida. Obtenidos los valores de salida de la red, se inicia la segunda fase, comparándose éstos valores con la salida esperada para así obtener el error. Se ajustan los pesos de la última capa proporcionalmente al error. Se pasa a la capa anterior con una retropopagación del error, ajustando los pesos y continuando con este proceso hasta llegar a la primera capa. De esta manera se han modificado los pesos de las conexiones de la red para cada patrón de aprendizaje del problema, del que conocíamos su valor de entrada y la salida deseada que debería generar la red ante dicho patrón.

La técnica Backpropagation requiere el uso de neuronas cuya función de activación sea continua, y por lo tanto, diferenciable. Generalmente, la función utilizada será del tipo sigmoidal.

3.4 Minería de Datos

La Minería de Datos (DM) por las siglas en inglés Data Mining es el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos (Frank 2000). Las herramientas de Data Mining predicen futuras tendencias y comportamientos, permitiendo la toma de decisiones.

Existen términos que se utilizan frecuentemente como sinónimos de la minería de datos. Uno de ellos se conoce como "análisis (inteligente) de datos" (Berthold y

Hand 2003), que suele hacer un mayor hincapié en las técnicas de análisis estadístico. Otro término muy utilizado, y el más relacionado con la minería de datos, es la extracción o "descubrimiento de conocimiento en bases de datos" (Knowledge Discovery in Databases o KDD, según sus siglas en inglés) (Orallo Hernández 2004).

Aunque algunos autores usan los términos minería de datos y KDD indistintamente, como sinónimos, existen claras diferencias entre los dos. Así la mayoría de los autores coinciden en referirse al KDD como un proceso que consta de un conjunto de fases, una de las cuales es la minería de datos. (Berthold y Hand 2003) De acuerdo con esto, el proceso de minería de datos consiste únicamente en la aplicación de un algoritmo para extraer patrones de datos y se llamará KDD al proceso completo que incluye pre-procesamiento, minería y post-procesamiento de los datos.

El KDD (Fayyad 1996) es la extracción automatizada de conocimiento o patrones interesantes, no triviales, implícitos, previamente desconocidos, potencialmente útiles y predictivos de la información de grandes Bases de Datos.

La Ilustración 23 muestra las fases del proceso de KDD, una de las cuales es la Minería de Datos

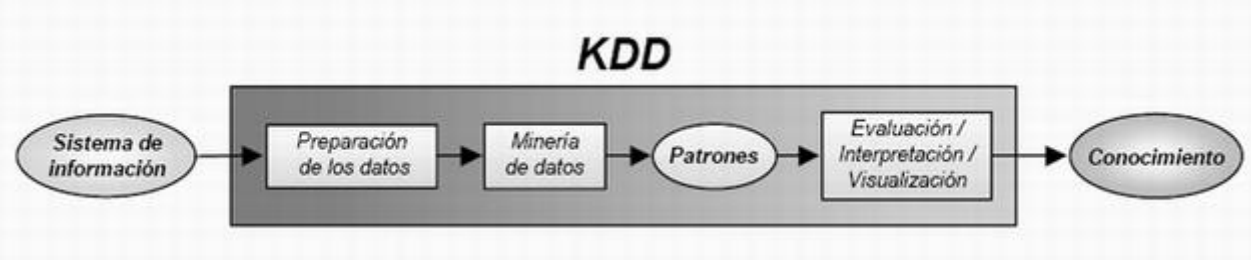


Ilustración 14 Fases del Proceso KDD (Ernesto González Díaz s.f.)

Las investigaciones en temas de KDD incluyen análisis estadístico, técnicas de representación del conocimiento y visualización de datos, entre otras. Algunas de las tareas más frecuentes en procesos de KDD son la clasificación y clustering, el

reconocimiento de patrones, las predicciones y la detección de dependencias o relaciones entre los datos (Ernesto González Díaz s.f.).

La minería de Datos utiliza el análisis matemático para deducir los patrones y tendencias que existen en los datos. Normalmente, estos patrones no se pueden detectar mediante la exploración tradicional de los datos porque las relaciones son demasiado complejas o porque hay demasiado datos[8].

Estos patrones y tendencias se pueden recopilar y definir como un *modelo de minería de datos*. Los modelos de minería de datos se pueden aplicar en escenarios como los siguientes:

- **Pronóstico**: cálculo de las ventas y predicción de las cargas del servidor o del tiempo de inactividad del servidor.
- **Riesgo y probabilidad**: elección de los mejores clientes para la distribución de correo directo, determinación del punto de equilibrio probable para los escenarios de riesgo, y asignación de probabilidades a diagnósticos y otros resultados.
- **Recomendaciones**: determinación de los productos que se pueden vender juntos y generación de recomendaciones.
- **Búsqueda de secuencias**: análisis de los artículos que los clientes han introducido en el carrito de la compra y predicción de posibles eventos.
- **Agrupación**: distribución de clientes o eventos en grupos de elementos relacionados, y análisis y predicción de afinidades.

La generación de un modelo de minería de datos forma parte de un proceso mayor que incluye desde la formulación de preguntas acerca de los datos y la creación de un modelo para responderlas, hasta la implementación del modelo en un entorno de trabajo. Este proceso se puede definir mediante los seis pasos básicos siguientes (Microsoft s.f.):

1. Definir el problema

[8] **Conceptos de minería de datos, https://msdn.microsoft.com/es-es/library/ms174949%28v=sql.120%29.aspx**

2. Preparar los datos
3. Explorar los datos
4. Generar modelos
5. Explorar y validar los modelos
6. Implementar y actualizar los modelos

El siguiente diagrama describe las relaciones existentes entre cada paso del proceso.

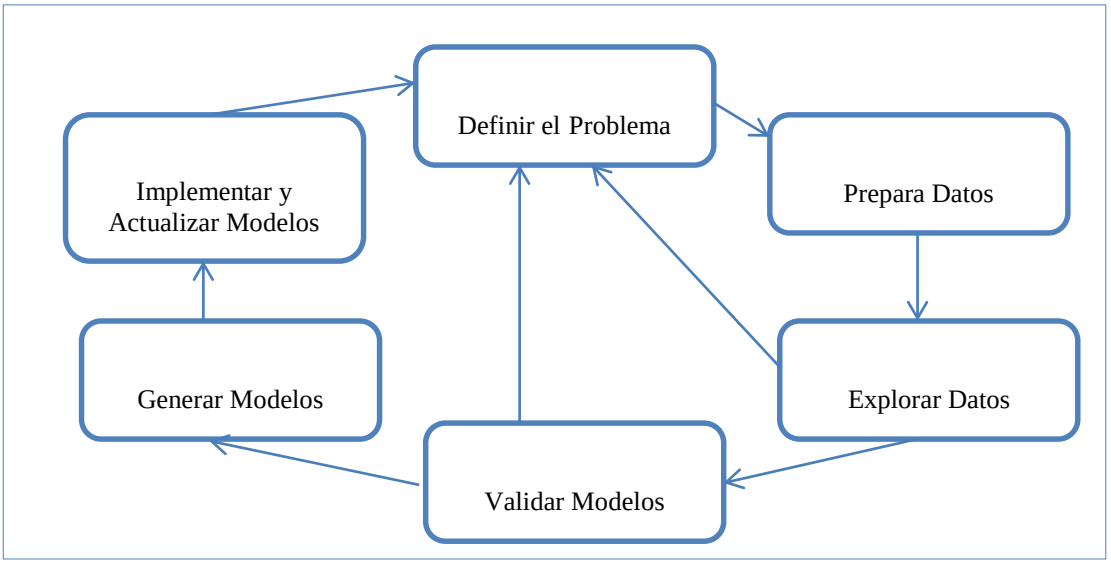


Ilustración 15 Implementación de un proceso de Minería de Datos (Microsoft s.f.)

3.4.1.1 Minería de Datos y Redes Neuronales

El descubrimiento de conocimiento en bases de datos de información científica puede ser entendido como un proceso que implica la realización de una secuencia básica de tareas:

1. Comprensión del campo de aplicación
2. Adquisición y selección de ficheros.

3. Preprocesamiento de ficheros.
4. Minería de Datos de resultados.
5. Visualización e interpretación de resultados.
6. Evaluación y reporte de resultados.

Este proceso de descubrimiento de conocimiento es iterativo e interactivo y tiene a la minería de datos como una de sus principales etapas (J. L. Sang 2001). La minería de datos integra métodos estadísticos con métodos de “aprendizaje automático”, y en particular redes neuronales, para llevar a cabo el proceso de análisis exploratorio de datos (Mannila s.f.). Contar con el conocimiento y la experiencia de los expertos es una parte importante del análisis. Es importante que estos participen activamente en el proceso de interpretación, visualización, evaluación y reporte de resultados.

Inspirados en la anatomía y fisiología del cerebro humano, las Redes Neuronales Artificiales (RNA) son modelos matemáticos que permiten hacer computación inteligente (A. K. Jain 1996) y llevar a cabo tareas que las computadoras seriales no pueden realizar: reconocimiento de patrones, memorias y aprendizaje asociativo, control adaptivo, predicción de series de tiempo, clasificación de señales y clustering, entre otras.

En una computadora neuronal el procesamiento es distribuido a toda una red de procesadores denominados “neuronas” que realizan el cómputo en paralelo. La propiedad de distribución y la capacidad de paralelizar los procesos determinan las nuevas capacidades implicadas en el paradigma neuronal. Desde el punto de vista de la minería de datos, el procesamiento paralelo 4 y distribuido es muy importante porque permite que las redes neuronales sean capaces de llevar a cabo el procesamiento de datos a una escala masiva (Bigus 1996).

Los modelos de minería de datos construidos con un algoritmo de red neuronal pueden contener varias redes, en función del número de columnas que se utilizan para la entrada y la predicción, o sólo para la predicción. El número de redes que

contiene un único modelo de minería de datos depende del número de estados que contienen las columnas de entrada y las columnas de predicción que utiliza el modelo.

Una vez procesado el modelo, puede usar la red y los pesos almacenados dentro de cada nodo para realizar predicciones. Un modelo de red neuronal admite el análisis de regresión, de asociación y de clasificación. Por lo tanto, el significado de cada predicción puede ser diferente. También puede consultar el propio modelo, revisar las correlaciones encontradas y recuperar las estadísticas relacionadas.

4. METODOLOGÍA Y HERRAMIENTAS DEL MODELO PROPUESTO

El poder de las redes neuronales recae en su habilidad para capturar relaciones no lineales inherentes en los datos, mientras que los modelos lineales describen una relación lineal entre observaciones actuales y posteriores, las redes neuronales describen una relación no lineal entre las dos. Debido a esto son de gran utilidad en el área de predicciones.

Según Dorffner (Dorffner 2008), el Perceptron Multicapa es el tipo de red neuronal artificial más utilizado y con los mejores resultados para este tipo de aplicación (predicciones). Se pretende crear un modelo para diseñar una red neuronal artificial para predicciones.

Se empieza identificando las variables de entrada, seguidamente se procede a la recopilación de datos asegurando su consistencia, luego se definen los conjuntos de entrenamiento, prueba y validación en base a los datos obtenidos. A partir de este momento es tiempo de comenzar a tomar en cuenta los paradigmas de diseño de la red neuronal artificial como el número de capas, el número de neuronas en cada capa y la función de transferencia. Luego debemos seleccionar los criterios de evaluación de los resultados que arroja la red para después planificar la fase de aprendizaje, teniendo en cuenta factores tales como el número de iteraciones y la tasa de aprendizaje. Por último se implementa el modelo.

4.1 División de Sistemas

Es una División adscrita a la Vicerrectoría Administrativa, responsable de la modernización tecnológica enfocada principalmente a tener una Universidad Tecnológica de Pereira con:

- Infraestructura informática actualizada de acuerdo a sus necesidades y disponible para la demanda de servicios de información
- Sistemas de información consolidado para la toma de decisiones, automatización de procesos y desarrollo de conocimiento
- Servicios Informáticos de calidad para beneficio de la comunidad

Área encargada del análisis, diseño y desarrollo de software específico requerido por los diferentes procesos de la Universidad, utilizando las diferentes herramientas y haciendo uso adecuado del BI (Inteligencia de negocios), con el fin de tener un sistema de información consolidado para toma de decisiones, automatización de procesos y desarrollo de conocimiento (CRIE s.f.).

4.2 Metodología de entrenamiento

La metodología para para la detección de errores en proyectos de software está compuesto por los siguientes pasos (Boyd 1996):

Paso 1: Selección de Variables
Paso 2: Recolección de Datos
Paso 3: Pre-procesamiento de Datos
Paso 4: Definición de Conjunto de Entrenamiento, Validación y Prueba
Paso 5: Selección de la Arquitectura de Redes Neuronales

a. Número de Neuronas de Entrada
b. Número de Capas Ocultas
c. Número de Neuronas Ocultas
d. Número de Neuronas de Salida

e. Función de Transferencia

Paso 6: Criterios de Evaluación

Paso 7: Entrenamiento de la Red Neuronal

a. Número de Iteraciones
b. Tasa de Aprendizaje y Momentum

Paso 8: Implementación del Modelo de Redes Neuronales.

4.3 Algoritmo de entrenamiento (backpropagation):

Paso 1. Inicializar los pesos de la red con valores pequeños aleatorios.

Paso 2. Presentar un patrón de entrada y especificar la salida deseada que debe generar la red.

Paso 3. Calcular la salida actual de la red. Para ello se presentan las entradas a la red y se calcula la salida de cada capa hasta llegar a la capa de salida, ésta será la salida de la red. Los pasos son los siguientes:

Se calculan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada. Para una neurona j oculta:

Ecuación 9 Entradas netas para las neuronas ocultas

$$net_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$

El índice h se refiere a magnitudes de la capa oculta; el subíndice p, al p-ésimo vector de entrenamiento, y j a la j-ésima neurona oculta. El término θ puede ser opcional, pues actúa como una entrada más.

Se calculan las salidas de las neuronas ocultas:

Ecuación 10 Salidas de las neuronas ocultas

$$y_{pj} = f_{j}^{h}(net_{pj}^{h})$$

Se realizan los mismos cálculos para obtener las salidas de las neuronas de salida:

Ecuación 11 Salidas de las neuronas de salida

$$net_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} y_{pj} + \theta_{k}^{0}$$
$$y_{pk} = f_{k}^{0}(net_{pk}^{o})$$

Paso 4. Calcular los términos de error para todas las neuronas.

Si la neurona k es una neurona de la capa de salida, el valor de la delta es:

Ecuación 12 Valor delta

$$\delta_{pk}^{0} = (d_{pk} - y_{pk}) f_{k}^{o'}(net_{pk}^{o})$$

La función f debe ser derivable. En general disponemos de dos formas de función de salida:

La función lineal:

Ecuación 13Función lineal

$$f_{k}(net_{jk}) = net_{jk}$$

La función sigmoidal:

Ecuación 14 Función sigmoidal

$$f_{k}(net_{jk}) = \frac{1}{1 + e^{-net_{jk}}}$$

La selección de la función depende de la forma que se decida representar la salida: si se desea que las neuronas de salida sean binarias, se utiliza la función sigmoidal, en otros casos, la lineal. Para una función lineal, tenemos: f_k^0*, mientras que la derivada de una función sigmoidal es:* $f_k^{0\prime}=f_k^0(1-f_k^0)=y_{pk}(1-y_{pk})$ *por lo que los términos de error para las neuronas de salida quedan:*

$\delta_{pk}^{o} = (d_{pk} - y_{pk})$ *Para la salida lineal.*

$\delta_{pk}^{o} = (d_{pk} - y_{pk}) y_{pk}(1 - y_{pk})$ *Para la salida sigmoidal.*

Si la neurona j no es de salida, entonces la derivada parcial del error no puede ser evaluada directamente, por tanto se obtiene el desarrollo a partir de valores que son conocidos y otros que pueden ser evaluados.
La expresión obtenida en este caso es:

Ecuación 15 Derivada parcial de error

$$\delta_{pj}^{h} = f_{j}^{h\prime}(net_{pj}^{h})\sum_{k}\delta_{pk}^{0} w_{kj}^{0}$$

donde observamos que el error en las capas ocultas depende de todos los términos de error de la capa de salida. De aquí surge el término propagación hacia atrás.

Paso 5. Actualización de los pesos: para ello utilizamos un algoritmo recursivo, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada, ajustando los pesos de la siguiente forma:

Para los pesos de las neuronas de la capa de salida:

Ecuación 16 Pesos de las neuronas de la capa de salida

$$w_{kj}^{0}(t+1) = w_{kj}^{o}(t) + \Delta w_{kj}^{o}(t+1)$$
$$\Delta w_{kj}^{o}(t+1) = \alpha\delta_{pk}^{o} y_{pj}$$

Para los pesos de las neuronas de la capa oculta:

Ecuación 17 Pesos de las neuronas de la capa oculta

$$w_{ji}^{h}(t+1) = w_{ji}^{h}(t) + \Delta w_{ji}^{h}(t+1)$$
$$\Delta w_{ji}^{h}(t+1) = \alpha\delta_{pj}^{h} x_{pi}$$

En ambos casos, para acelerar el proceso de aprendizaje se puede añadir un término momento.

Paso 6. El proceso se repite hasta que el término de error,

Ecuación 18 Término de error

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2$$

resulta aceptablemente pequeño para cada uno de los patrones aprendidos. (Martinez 2000) (Skapura 1993) (Sanz 2002)

4.4 Tipo de Investigación

El proyecto está referido a una investigación experimental y descriptiva. Se considera experimental dada la existencia de una variable independiente y otra dependiente, generando de esta forma una alteración en la variable dependiente. Es descriptiva dado que se miden los indicadores de la variable dependiente.

4.5 Técnica

Redes Neuronales Artificiales.
Técnica de Escalamiento.

4.6 Herramienta

Software de Redes Neuronales desarrolladas por la autora en lenguaje Delphi 7.0.
Consta de Tres Módulos Principales:

1. Configuración y Creación de la Red: en este módulo podemos configurar el número de neuronas de cada una de las capas (entrada, oculta y salida).
   La red solo maneja tres capas, una de entrada una oculta y otra de salida.
   Los pesos pueden ser asignados o creados aleatoriamente.

2. Entrenamiento y Validación de la Red: Luego de la configuración de la Red Neuronal podemos comenzar con el proceso de entrenamiento.
3. Predicción: Luego de haber creado y entrenado el modelo pasamos a utilizarlo en las predicciones.

4.7 Fuente

Los datos son tomados de software desarrollado por la División de Sistemas de la Universidad Tecnológica de Pereira desde el 01 de enero de 2008 hasta el 31 de diciembre de 2013.
Los módulos a analizar son los siguientes:

Tabla 2 Módulos a Analizar

Aplicativos	**Módulos**
Administración de Programas Académicos	**3**
Administración de Asignaturas / Planes de Estudios Gráfico	**4**
Calendario Académico	**1**
Administración de Usuarios	**1**
Portal Estudiantil	**1**
Portal Docente	**1**
Configuración de Inscripciones	**3**
Administrador de Pines	**1**
Inscripciones Web	**3**
Admisiones	**3**
Administrador de Inscripciones	**1**
Programación de Horario de Asignaturas	**2**
Administración de Grupos Temporales	**1**
Asignación Automática de Horarios Estudiantes	**4**
Asignación Docente a Grupos y Contratación	**2**

Liquidación Financiera	**3**
Subir Archivos de Pago de Matriculas	**2**
Módulo de Pagos Electrónicos	**2**
Matricula Web Estudiantes	**3**
Matricula Registro	**2**
Hoja de Vida Personal	**3**
Hechos Académicos	**1**
Registro Estudiantil	**3**
Registro de Notas por parte Docente	**4**
Consulta de Información (Decanos)	**3**
Administración de Graduaciones	**1**
Participantes a las Graduaciones	**2**
Administrador de Certificados	**1**
Administrador permisos de notas extemporáneas	**1**
Sistema de Documentos digitales	**3**
Solicitudes web	**3**
Configuración de solicitudes	**1**
Auditorias	**2**
Administrador y Asignación de aulas	**3**
Total Registros	**74**

5. DISEÑO Y ENTRENAMIENTO DE LA RED NEURONAL DEL MODELO PROPUESTO

A continuación se mostrará cómo fue diseñada la red neuronal con las especificaciones del modelo, su estructura y los algoritmos y reglas utilizados para entrenarla.

5.1 Diseño de Modelo de RNAs

5.1.1 Selección de Variables

Determinar que variables de entrada son importantes es un asunto crítico. En este caso, el interés en la predicción involucra emplear como entradas datos técnicos manejados por la división de sistemas.

- **Variables de Salida:**
    1. ***Falla de seguridad:*** Es cuando se puede violar los protocolos de acceso al software o cuando los datos no quedan protegido debidamente y pueden ser hurtados o modificados sin autorización.
       Tipo de Variable: cuantitativa.
       Rango: 0 o 1.
    2. ***Falla de carga:*** Cada vez que un usuario ingresa a una aplicación se crea una instancia de esta, lo que ocupa memoria y espacio en los servidores, muchas veces el espacio asignado a cada aplicativo es menor a que realmente este ocupa cuando empieza a funcionar (por lo general es debido al número de usuarios) lo que produce fallas en el software.
       Tipo de Variable: cuantitativa.
       Rango: 0 o 1.
    3. ***Falla de Funcionalidad:*** Es cuando un software no tiene la capacidad funcional para satisfacer las necesidades explicitas e implícitas bajo funcionamiento.
       Tipo de Variable: cuantitativa.

Rango: 0 o 1.

En las tres variables de salida los valores de 1 indican error de ese tipo de variable y los valores de 0 indican que no hay errores de ese tipo.

- **Variables de Entrada:**
  1. ***Funcionabilidad*** (Yaidelyn Macías Rivero 2009)**:** Es la capacidad del producto de software para proveer las funciones que satisfacen las necesidades explícitas e implícitas cuando este se utiliza bajo condiciones específicas.

     Capacidad de la interfaz visual: capacidad del producto de software para mostrar los resultados del modo más legible posible.

     Informes, estadísticas: capacidad del producto de software para ofrecer los informes y estadísticas de la manera más precisa posible según la necesidad.

     Seguridad: se refiere a la habilidad de prevenir el acceso no autorizado, sea accidental o premeditado, a los programas y datos.

  2. ***Facilidad de mantenimiento*.** El mantenimiento del software cuenta con más esfuerzo que cualquier otra actividad de ingeniería del software. La facilidad de mantenimiento es la habilidad con la que se puede corregir un programa si se encuentra un error, se puede adaptar si su entorno cambia u optimizar si el cliente desea un cambio de requisitos. No hay forma de medir directamente la facilidad de mantenimiento; por consiguiente, se deben utilizar medidas indirectas. Una métrica orientada al tiempo simple es el *tiempo medio de cambio* (TMC), es decir, el tiempo que se tarda en analizar la petición de cambio, en diseñar una modificación

apropiada, en efectuar el cambio, en probarlo y en distribuir el cambio a todos los usuarios. En promedio, los programas que son más fáciles de mantener tendrán un TMC más bajo (para tipos equivalentes de cambios) que los programas que son más difíciles de mantener (Aguirre 2012).

3. ***Integridad.*** En esta época de intrusos informáticos y de virus, la integridad del software ha llegado a tener mucha importancia. Este atributo mide la habilidad de un sistema para soportar ataques (tanto accidentales como intencionados) contra su seguridad. El ataque se puede ejecutar en cualquiera de los tres componentes del software, ya sea en los programas, datos o documentos.

   Para medir la integridad, se tienen que definir dos atributos adicionales: amenaza y seguridad. La a*menaza* es la probabilidad (que se logra evaluar o concluir de la evidencia empírica) de que un ataque de un tipo establecido ocurra en un tiempo establecido. La *seguridad* es la probabilidad (que se puede estimar o deducir de la evidencia empírica) de que se pueda repeler el ataque de un tipo establecido, en donde la integridad del sistema se puede especificar como (Aguirre 2012):

   Integridad = [1-amenaza*(1-seguridad)]

4. ***Portabilidad*** (Yaidelyn Macías Rivero 2009)**:**
   - *Facilidad de instalación*: capacidad del software para ser instalado en un ambiente específico.
   - *Adaptabilidad*: capacidad del software para ser adaptado a diferentes entornos especificados sin aplicar acciones o medios diferentes de los previstos para el propósito del software considerado.

- *Coexistencia*: capacidad del software para coexistir con otros productos de software independientes dentro de un mismo entorno, compartiendo recursos comunes.
- R*eemplazabilidad*: capacidad del software para ser utilizado en lugar de otro producto de software, para el mismo propósito y en el mismo entorno.

5. ***Facilidad de uso.*** El calificativo “amigable con el usuario” se ha transformado universalmente en disputas sobre productos de software. Si un programa no es “amigable con el usuario”, prácticamente está próximo al fracaso, incluso aunque las funciones que realice sean valiosas. La facilidad de uso es un intento de cuantificar “lo amigable que pude ser con el usuario” y se consigue medir en función de cuatro características (Aguirre 2012):
   - Destreza intelectual y/o física solicitada para aprender el sistema.
   - El tiempo requerido para alcanzar a ser moderadamente eficiente en el uso del sistema.
   - Aumento neto en productividad (sobre el enfoque que el sistema reemplaza) medida cuando alguien emplea el sistema moderadamente y eficientemente.
   - Valoración subjetiva (a veces obtenida mediante un cuestionario) de la disposición de los usuarios hacia el sistema.

6. **Fiabilidad:** Se refiere a la capacidad del software de mantener su nivel de ejecución bajo condiciones normales en un período de tiempo establecido.
   Variables a tener en cuenta:

- *Tolerancia a fallas*: se refiere a la habilidad de mantener un nivel específico de funcionamiento en caso de fallas del software o en caso de ocurrencia de infracciones de su interfaz específica.
- *Recuperación*: se refiere a la capacidad de restablecer el nivel de operación y recobrar los datos que fueron afectados directamente por una falla, así como el tiempo y el esfuerzo necesarios para lograrlo (Yaidelyn Macías Rivero 2009).

Todos los fallos del software, se producen por problemas de diseño o de implementación. Considerando un sistema basado en computadora, una medida sencilla de la fiabilidad es el *tiempo medio entre fallos* (TMEF) (Mayrhauser 1991), donde:

TMEF = TMDF+TMDR

(TMDF (*tiempo medio de fallo*) y TMDR (*tiempo medio de reparación*))

Muchos investigadores argumentan que el TMDF es una medida más útil que los defectos/KLDC, simplemente porque el usuario final se enfrenta a los fallos, no al número total de errores. Como cada error de un programa no tiene la misma tasa de fallo, la cuenta total de errores no es una buena indicación de la fiabilidad de un sistema.

7. **Disponibilidad:** Es la probabilidad de que un programa funcione de acuerdo con los requisitos en un momento dado, y se define como:

Disponibilidad = TMDF/(TMDF + TMDR) x 100 %

La medida de fiabilidad TMEF es igualmente sensible al TMDF que al TMDR. La medida de disponibilidad es algo más sensible al TMDR

ya que es una medida indirecta de la facilidad de mantenimiento del software.

8. **Efectividad en la eliminación de defectos (EED):** es una medida de la habilidad de filtrar las actividades de la garantía de calidad y de control al aplicarse a todas las actividades del marco de trabajo del proceso. Cuando se toma en consideración globalmente para un proyecto, EED se define de la forma siguiente:

$$EED = E / (E + D)$$

donde E= es el número de errores encontrados antes de la entrega del software al usuario final y D= es el número de defectos encontrados después de la entrega. El valor ideal de EED es 1, donde simbolizando que no se han encontrado defectos en el software. De forma realista, *D* será mayor que cero, pero el valor de EED todavía se puede aproximar a 1 cuando *E* aumenta. En consecuencia cuando *E* aumenta es probable que el valor final de D disminuya (los errores se filtran antes de que se conviertan en defectos) Si se utiliza como una métrica que suministra un indicador de la destreza de filtrar las actividades de la garantía de la calidad y el control, el EED alienta a que el equipo del proyecto de software instituya técnicas para encontrar los errores posibles antes de su entrega.

Del mismo modo el EED se puede manipular dentro del proyecto, para evaluar la habilidad de un equipo en encontrar errores antes de que pasen a la siguiente actividad, estructura o tarea de ingeniería del software. Por ejemplo, la tarea de análisis de requerimientos produce un modelo de análisis qué se puede inspeccionar para encontrar y corregir errores. Esos errores que no se encuentren durante la revisión del modelo de análisis se pasan a las tareas de

diseño (en donde se puede encontrar o no) Cuando se utilizan en este contexto, el EED se vuelve a definir como:

$$EED = Ei / (Ei + Ei+1)$$

Donde E*i* = es el número de errores encontrados durante la actividad *iesima* de: ingeniería del software *i*, el Ei + 1 = es el número de errores encontrado durante la actividad de ingeniería del software (i + 1) que se puede seguir para llegar a errores que no se detectaron en la actividad *i.*

Un objetivo de calidad de un equipo de ingeniería de software es alcanzar un EED que se aproxime a 1, esto es, los errores se deberían filtrar antes de pasarse a la actividad siguiente. Esto también puede ser utilizado dentro del proyecto para evaluar la habilidad de un equipo, esto con el objetivo de encontrar deficiencias que harán que se atrase el proyecto (Aguirre 2012).

9. **Satisfacción:** La capacidad del software para satisfacer a los usuarios en un contexto especificado de uso. La satisfacción es la respuesta del usuario a la interacción con el producto, e incluye las actitudes hacia el uso del producto (Yaidelyn Macías Rivero 2009).

5.1.2 Escalamiento de Datos

Los datos de entrada son numéricos y en muchos casos puede ser conveniente escalarlos y/o preprocesarlos. El escalamiento se refiere a un cambio de escala, convertirlos a datos entre 0 y 1, entre -1 y 1, o estandarizarlos, de acuerdo al rango de las funciones de activación involucradas (Duarte 2001).

Para el proyecto los transformaremos a valores comprendidos entre 0 y 1, utilizando la siguiente formula:

Ecuación 19 Fórmula de escalamiento de datos (Duarte 2001)

$$z_t = \frac{y_t - \text{Min}}{\text{Max} - \text{Min}}$$

Dónde:

y_t : son los valores originales del dato de entrada.

Min: valor mínimos del dato de entrada

Max: valor máximo del dato de entrada

z_t : dato de entrada transformado entre 0 y 1.

5.1.3 Recolección de Datos

Los datos utilizados para este trabajo fueron entregados por parte de la División de Sistemas de la Universidad Tecnológica de Pereira correspondiente a los módulos desarrollados por dicha división desde el 01 de enero de 2008 hasta el 31 de diciembre de 2013.

5.1.4 Pre-procesamiento de Datos

El pre-procesamiento de datos se refiere al análisis y la transformación de variables de entrada y salida para minimizar el ruido, enfatizar las relaciones, detectar tendencias, y aplanar la distribución de la variable para ayudar a la red neuronal en el aprendizaje de patrones relevantes. Como las redes neuronales con buscadores de patrones, la representación de los datos es crítica en el diseño de una red exitosa.

La etapa de procesamiento no fue necesaria para el modelo ya que los datos no contienen valores nulos inconsistentes. Sin embargo, el software de implementación tiene funciones de pre-procesamiento y pos-procesamiento implícitas en las funciones de entrenamiento.

5.1.5 Definición de Conjunto de Entrenamiento, Validación y Prueba

- Conjunto de Entrenamiento: es el grupo de datos que le sirve a la red neuronal para aprender los patrones presentes en los datos, es decir la obtención de los pesos de la red. En este caso este grupo representa un total de 74 registros.
- Conjunto de Validación: es utilizado para la comprobación final de la red, donde los datos empleados son los más recientes consecuentemente al último valor de la muestra. El conjunto de validación debe consistir en las más recientes y contiguas observaciones. Se debe procurar no usar el conjunto de validación como conjunto de testeo tras ejecutar repetidamente pasos de series de entrenamiento-testeo-validación y adaptar las variables de entrada basadas en la actuación de la red en el conjunto de validación.
- Conjunto de Prueba: Corresponde a los datos restantes, una vez seleccionados los patrones de entrenamiento. Este conjunto de datos se utiliza para evaluar la precisión de la red. Está compuesto de los módulos desarrollados desde enero de 2014 hasta 30 de septiembre del mismo año.

5.2 Selección de la arquitectura de redes neuronales

Utilizaremos el algoritmo de entrenamiento Backpropagation. Modelaremos una red multicapa con retroalimentación. Su diseño básico es bastante simple aunque de gran potencia. El uso de redes neuronales para la predicción se está haciendo bastante popular debido a que da un enfoque y aproximación fácil para resolver problemas complejos. El algoritmo de retro-propagación o propagación hacia atrás permite una transmisión de los errores de la salida hacia atrás, hacia todas las neuronas de forma que se van ajustando los pesos de las conexiones con la idea de que el error disminuya. El algoritmo de aprendizaje está basado en la regla delta. De esta forma la función o superficie de error que tenemos asociada a la red

se va modificando según vamos cambiando el tamaño de los pesos entre las conexiones neuronales, hasta que se consigue minimizar el error de la superficie.
La principal ventaja de la Backpropagation es su capacidad genérica de mapeo de patrones. La red es capaz de aprender una gran variedad de relaciones de mapeo de patrones. No requiere un conocimiento matemático de la función que relaciona los patrones de la entrada y los patrones de salida. La Backpropagation sólo necesita ejemplos de mapeo para aprender. La flexibilidad de esta red es aumentada con la posibilidad de elegir número de capas, interconexiones, unidades procesadoras, constante de aprendizaje y representación de datos. Como resultado de estas características la red Backpropagation es capaz de participar con éxito en una amplia gama de aplicaciones (Olabe s.f.).

5.2.1 Número de neuronas de entrada

Corresponde a cada una de las variables de entrada, es decir 9 neuronas.

5.2.2 Número de Capas Ocultas

Las capas ocultas dan a la red la habilidad de generalizar, y en la práctica las redes neuronales con una o dos capas ocultas son las más utilizadas y han tenido un buen desempeño (Boyd 1996).El incremento en el número de capas también incrementa el tiempo de procesamiento y el peligro de sobre ajuste lo que conduce a un pobre desempeño en la predicción fuera de muestra. El sobre ajuste ocurre cuando un modelo de predicción tiene muy pocos grados de libertad. En otras palabras, se tienen relativamente pocas observaciones en relación con sus parámetros y por lo tanto es capaz de memorizar datos individuales en lugar de aprender patrones generales.
Es por eso que se recomienda que todas las redes neuronales comiencen de preferencia con una o a lo mucho con dos capas.
Hemos optado por una sola capa oculta la cual es suficiente para asegurar la capacidad de generalización de la red dada la cantidad de datos con los que contamos. Esto da un total de 3 capas, la capa de entrada, la capa oculta y la capa de salida.

5.2.3 Número de Neuronas Ocultas

El número de neuronas de la capa oculta será de un 75% del total de entradas (Boyd 1996). Es decir, dado que el número de entradas de la red es 9, el número de neuronas de la capa oculta sería de 7 neuronas.

5.2.4 Número de Neuronas de Salida

Decidir el número de neuronas de salida es algo más simple ya que hay muchas razones para emplear una sola neurona de salida. Las redes neuronales con múltiples salidas, especialmente si éstas salidas están ampliamente espaciadas, producirán resultados inferiores en comparación con una red con una única salida. Lo recomendable es tener una red especializada para cada una de las salidas deseadas en cada predicción.

El número de neuronas de la capa de salida de la red es tres, correspondiente a cada una de las variables de salida especificadas en la sección 5.1.1.

5.2.5 Función de Transferencia

El propósito de esta función es prevenir a las salidas de alcanzar valores muy elevados los cuales pueden paralizar la red y detener el entrenamiento de la misma. Como función de transferencia usaremos la función sigmoidal (cuyo rango de salida se encuentra entre -1 y +1) la cual es preferente para tareas de predicción según (Boyd 1996).

5.2.6 Conexiones

En el *entrenamiento* de una red neuronal tanto el *peso sináptico* de las conexiones como el *valor umbral* para cada neurona se modifican (según un algoritmo de aprendizaje), con el fin de que los resultados generados por la red *coincidan* con (o se aproximen a) los resultados esperados.

Y para simplificar el sistema de entrenamiento, el valor *umbral* (*U*) pasa a expresarse como un peso sináptico más (-W_0), pero asociado a una *neurona siempre activa* (X_0). Esta *neurona siempre activa*, se denomina "*bias*", y se sitúa en la capa anterior a la neurona *Y*, tal como se muestra en la **ilustración 36**.

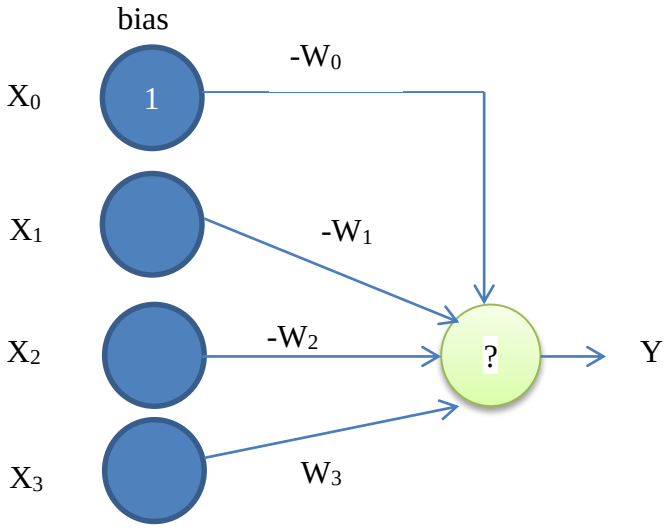


Ilustración 16 Neurona bias y su peso sináptico asociado (Martinez 2000)

Así, la condición de activación puede reescribirse como:

$$X_0W_0+X_1W_1+\ldots+X_iW_i>0 \leftrightarrow Activación \leftrightarrow Y=1$$

De esta manera el algoritmo de aprendizaje puede ajustar el *umbral* como si ajustara un *peso sináptico* más.

5.2.7 Criterios de Evaluación

En cuanto a los criterios de evaluación para medir la eficiencia de la red, se ha considerado utilizar el Error Cuadrático Medio (ECM). El Error Cuadrático Medio calculado como la diferencia entre la salida de la red y la respuesta deseada. El cual se usa como factor de culminación del entrenamiento. Para esta fase se fijó un parámetro en 2000^{10} épocas y el factor de terminación empleado del ECM teniendo como umbral un valor de 0.01.

5.3 Entrenamiento de la Red Neuronal

Entrenar una red neuronal para aprender patrones involucra el presentarle ejemplos de manera iterativa de las respuestas correctas. El objetivo del entrenamiento es encontrar un conjunto de pesos entre las neuronas que determinan el mínimo global de la función de error. A menos que el modelo esté sobre-ajustado, el conjunto de pesos debería proporcionar una buena generalización.

El entrenamiento de la red utiliza la técnica del gradiente descendente inmerso en el algoritmo de retropropagación. El entrenamiento se detendrá básicamente cuando suceda uno de estos 3 motivos: el número de iteraciones excede el número de épocas establecidas, si la función de evaluación (ECM) cae por debajo de la meta establecida, o si el error medio por la función de evaluación se incrementa para un número específico de iteraciones (este último caso requiere de la existencia del conjunto de validación). En cualquier caso los pesos y bias obtenidos son los encontrados en el mínimo error medido por la función de evaluación.

5.3.1 Número de Iteraciones (Épocas) Entrenamiento

Muchas veces el procedimiento de agregar un número mayor de iteraciones en el entrenamiento producirá un menor error en el entrenamiento, pero éste a su vez no garantiza que se obtendrá el menor error con el conjunto de prueba. E número máximo fue de 2000[10] épocas que fue suficiente para conseguir un nivel de error aceptable.

5.3.2 Tasa de Aprendizaje y Momento

La tasa de aprendizaje es una constante de proporcionalidad que determina el tamaño del cambio de los pesos. El cambio de los pesos de una neurona es

proporcional al impacto del peso de las neuronas sobre el error. Un método de incrementar la tasa de aprendizaje y por lo tanto agilizar el tiempo de entrenamiento sin caer en oscilaciones, es incluir un término de momento en la regla de aprendizaje de BP. El término de momento determina cómo los cambios pasados de los pesos afectan a los actuales cambios en los pesos. Este término suprime oscilaciones lado a lado filtrando variaciones de alta frecuencia. Cada nueva dirección de búsqueda es una suma ponderada de los gradientes actuales y previos.

5.4 Implementación del modelo de RNA

Se lleva a cabo la implementación del modelo haciendo uso de la técnica de Redes Neuronales Artificiales con un software creado por la autora desarrollada en Delphi 7.0.

Tabla 3 Resumen de Datos de la Red Neuronal

Resumen de Datos de la Red Neuronal	
Data Pre-procesada	74
Tipo de Variable	Los datos de entrada como de salida son de naturaleza analógica (valores reales continuos y normalmente normalizados, por lo que su valor absoluto será menor que la unidad). En este caso las funciones de activación de las neuronas serán también continuas, del tipo lineal o sigmoidal.
Error permitido de convergencia	0.01

Numero de Épocas	2000[10]
Tasa de Aprendizaje(α)	0.01
Número de Neuronas de Entrada	9
Número de Capas Ocultas	1
Número de Neuronas Ocultas en cada Capa	7
Número de Neuronas de Salida	3
Función de Transferencia	Función Sigmoidal
Momento	0.95
Función de Error	Error Cuadrático Medio

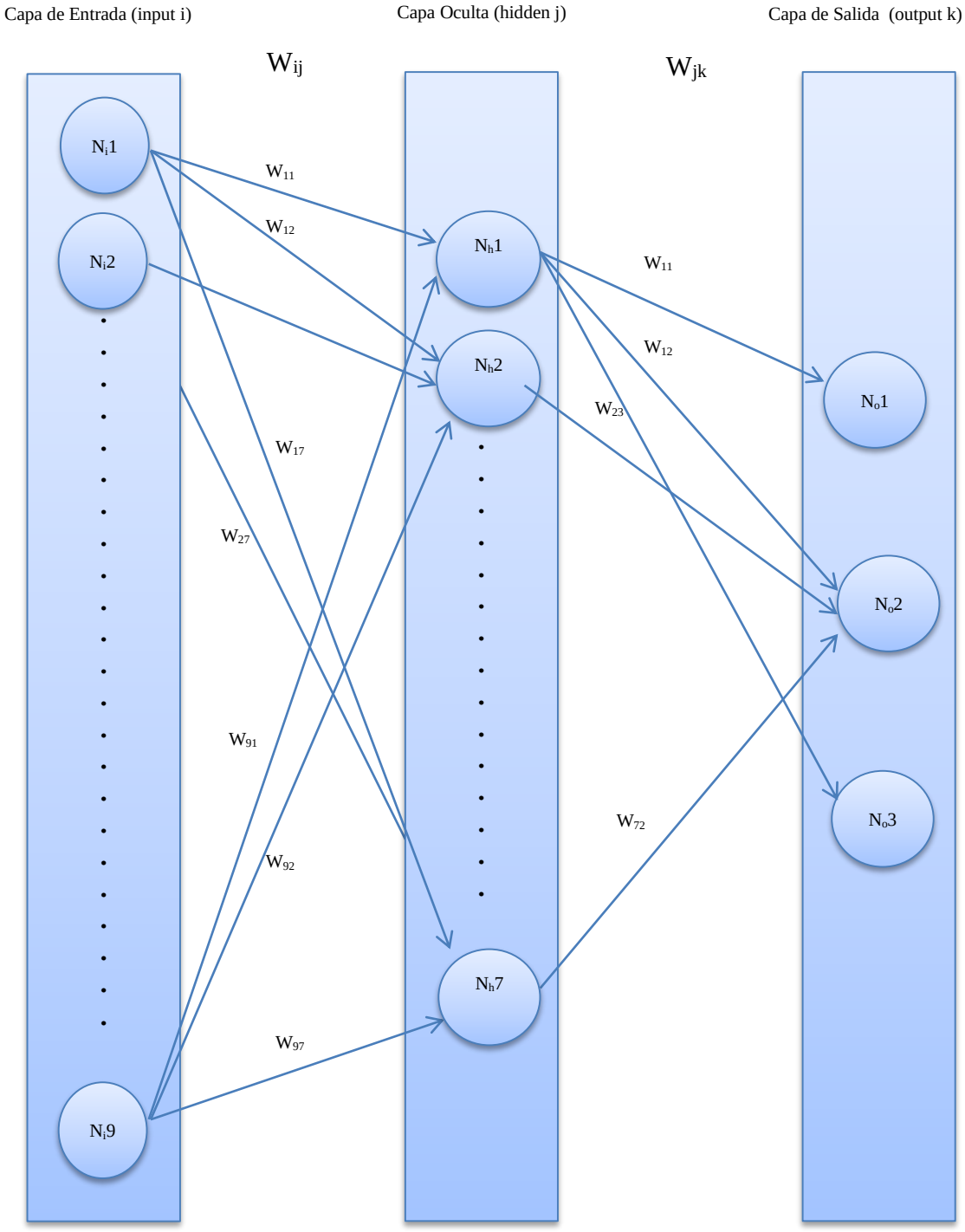


Ilustración 17 Arquitectura de La Red Neuronal

5.4.1 Valores de entrada y salida:

Se tiene 9 neuronas de entrada que representas las siguientes variables:

- *Funcionabilidad*
- *Facilidad de mantenimiento*
- *Integridad*
- *Portabilidad*
- *Facilidad de uso*
- Fiabilidad
- Disponibilidad
- Efectividad en la eliminación de defectos (EED)
- Satisfacción

Estos valores debido a que manejan diferentes rangos y valores que no se mueven entre cero y uno, se escalaron con la función de escalamiento mencionada en el apartado 5.1.2.

Los valores de las variables de salida en la etapa de entrenamiento fueron de cero y uno ya que son los valores esperados (falla de seguridad, falla de carga y falla de funcionalidad).

6. PRUEBA DEL MODELO

Después de entrenada la Red Neuronal se procede a probar el modelo con los datos de validación y prueba mencionados en la sección 5.1.5:

Tabla 4 Listado de Módulos para probar el modelo

Aplicativos	**Módulos**
Novedades de Nómina	**4**
Proveedores	**2**
Habeas Data	**2**
Total Registros	**8**

Tabla 5 Valores de Entrada (dados a la Red Neuronal)

Valores de Entrada(dados a la red neuronal)								
	Novedades de Nómina				***Proveedores***		***Habeas Data***	
	Mód 1	**Mód 2**	**Mód 3**	**Mód 4**	**Mód 1**	**Mód 2**	**Mód 1**	**Mód 2**
Funcionabilidad	**0.3**	**0.9**	**0.1**	**0.4**	**0.3**	**0.3**	**0.3**	**0.3**
Facilidad de mantenimiento	**0.9**	**0.8**	**0.7**	**0.9**	**0.4**	**0.3**	**0.2**	**0.5**
Integridad	**0.1**	**0.7**	**0.2**	**0.5**	**0.6**	**0.7**	**0.7**	**0.7**
Portabilidad	**0.4**	**0.6**	**0.4**	**0.5**	**0.4**	**0.4**	**0.8**	**0.4**
Facilidad de uso	**0.6**	**0.8**	**0.6**	**0.7**	**0.6**	**0.6**	**0.6**	**0.6**
Fiabilidad	**0.7**	**0.9**	**0.7**	**0.9**	**0.7**	**0.7**	**0.7**	**0.7**
Disponibilidad	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**
Efectividad en la eliminación de defectos (EED)	**0.9**	**1**	**0.9**	**0.8**	**0.9**	**1**	**0**	**0.9**
Satisfacción	**1**	**0.7**	**0**	**0.3**	**0.4**	**0.8**	**0.9**	**1**

Tabla 6 Valores de Salida (entregadas por la Red Neuronal)

Valores de Salida(entregadas por la Red Neuronal)								
	Novedades de Nómina				***Proveedores***		***Habeas Data***	
	Mód 1	**Mód 2**	**Mód 3**	**Mód 4**	**Mód 1**	**Mód 2**	**Mód 1**	**Mód 2**
falla de seguridad	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
falla de carga	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
falla de funcionalidad	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

7. ANALISIS DE RESULTADOS

La red tiene un método de entrenamiento supervisado donde se le presentaron parejas de patrones, un patrón de entrada emparejado con un patrón de salida deseada. Por cada presentación, los pesos fueron ajustados de tal forma que el error entre la salida deseada y la respuesta de la red disminuía.

El algoritmo de aprendizaje backpropagation conlleva una fase de propagación hacia adelante y otra fase de propagación hacia atrás. Ambas fases se realizaron por cada patrón presentado en la sesión de entrenamiento.

En la fase de propagación hacia adelante se presentó un patrón en la capa de entrada de la red. Cada unidad de entrada se correspondía con un elemento del vector patrón de entrada (variables de entrada). Las unidades de entrada toman el valor de su correspondiente elemento y se calculó el valor de activación o nivel de salida de la primera capa. A continuación la siguiente capa realizó la fase de propagación hacia adelante que determinó el nivel de activación de la capa de salida.

Conviene indicar que las unidades procesadoras de la capa de entrada no realizaron ninguna operación de cálculo con sus entradas, ni operaciones con

funciones umbrales, sólo asumieron su salida como el valor del correspondiente elemento del vector de entrada.

En el modelo de red se utilizaron unidades llamadas bias como parte de la capa oculta y la capa de entrada. Estas unidades presentaron constantemente un nivel de activación de valor 1. La utilización de esta unidad tenía un doble objetivo, mejorar las propiedades de convergencia de la red y ofrecer un nuevo efecto umbral sobre la unidad que opera.

Una vez completada la fase de propagación hacia adelante se inició la fase de corrección o fase de propagación hacia atrás. Los cálculos de las modificaciones de todos los pesos de las conexiones empezaron por la capa de salida y continuaron hacia atrás a través de todas las capas de la red hasta la capa de entrada.
El ajuste de estos pesos fue relativamente sencillo debido a que existe y se conocía el valor deseado para cada una de las unidades de la capa de salida. Cada unidad de la capa de salida produjo un número real como salida y se comparó con el valor deseado especificado en el patrón del conjunto de entrenamiento.

Las capas ocultas no tienen un vector de salida deseada y por tanto no se siguió el método de propagación de error en las unidades procesadoras de la capa de salida.

“La convergencia es un proceso en el que el valor ECM del error de la red tiende cada vez más al valor 0. La convergencia no siempre es fácil de conseguirla porque a veces el proceso puede requerir un tiempo excesivo o bien porque la red alcanza un mínimo local y deja de aprender”[9].

[9] Curso: Redes Neuronales Artificiales y sus Aplicaciones, Xabier Basogain Olabe.

Se debe resaltar el hecho que las redes neuronales fueron puestas a prueba con datos desconocidos o nuevos, por lo que las funciones de transferencia de las neuronas de salida, no entregó exactamente los valores de uno o cero sino que nos entregó una aproximación de esos valores, es decir, que entregó valores decimales entre cero y uno.

Los resultados que se obtienen son bastante satisfactorios dado que el Error Absoluto Medio Porcentual al evaluar la mejor configuración de red neuronal es inferior al 3%, existiendo una correlación entre la serie de datos entregados por la red y los esperados de 0,949.

Al variar la cantidad de datos utilizados en el entrenamiento desde un 80% a un 60% los resultados obtenidos no se ven afectados siendo capaz de generalizar en un conjunto de datos donde existe mayor probabilidad de encontrar patrones desconocidos. Al aplicar una distorsión aleatoria entre [-5%, +5%] a los datos de validación se produce un aumento del Error Absoluto Medio Porcentual de menos del 1%, lo cual demuestra la capacidad que tienen las redes neuronales de tolerar ruido en los datos.

Una vez finalizada la fase de aprendizaje la red pudo ser utilizada para lo que fue entrenada.

8. COMPARACIÓN DE RESULTADOS

En la práctica los módulos con los que se evaluó el modelo están en funcionamiento y algunos de ellos presentaron fallas de seguridad y de carga debido a un cambio en la plataforma de desarrollo y en la configuración de los servicios web. Cabe anotar que estas fallas todavía se presentan debido a que la plataforma está siendo configurada y adaptada a las necesidades de la División de Sistemas.

Los módulos que presentaron problemas fueron:

Tabla 7 Módulos que presentaron problemas

	Novedades de Nómina				***Proveedores***		***Habeas Data***	
	Mód 1	**Mód 2**	**Mód 3**	**Mód 4**	**Mód 1**	**Mód 2**	**Mód 1**	**Mód 2**
falla de seguridad	**X**	**X**	**X**					**X**
falla de carga		**X**				**X**		
falla de funcionalidad								

El modelo previó problemas en los siguientes módulos:

Tabla 8 Fallas previstas por el modelo

Valores de Salida(entregadas por la Red Neuronal)								
	Novedades de Nómina				***Proveedores***		***Habeas Data***	
	Mód 1	**Mód 2**	**Mód 3**	**Mód 4**	**Mód 1**	**Mód 2**	**Mód 1**	**Mód 2**
falla de seguridad			**X**					**X**
falla de carga		**X**				**X**		
falla de funcionalidad								

Como lo explicamos al inicio, el cambio en la herramienta de desarrollo por parte de la División de Sistemas no previó las fallas de seguridad que presentaron los módulos 1 y 2 del software de novedades de nómina, ya que dicho software está desarrollado en otra tecnología y con una arquitectura que todavía está en construcción, lo que hace que se presenten estas fallas de seguridad.

En cambio los aplicativos de Proveedores y Habeas Data están construidos en la plataforma habitual, razón por la cual el modelo pudo predecir correctamente las fallas en los módulos 2 de ambas aplicaciones.

9. CONCLUSIONES

Los resultados obtenidos comprobaron la hipótesis de la investigación, la cual postulaba que las Redes Neuronales podían ser una herramienta eficaz de estimación de calidad de las diferentes versiones de los proyectos de software de la Universidad Tecnológica de Pereira.

Como parte del objetivo general de esta investigación se ha evaluado la capacidad que tienen las redes neuronales artificiales en la predicción de calidad de software, resultando efectivas en esta tarea y demostrando que son una herramienta útil de predicción.

Uno de los principales inconvenientes en la utilización de redes neuronales es la elección de cada elemento de la arquitectura de red, al igual que los valores de entrada ya que los datos de los módulos elegidos para entrenar la red debían cumplir con las especificaciones estrictas de la metodología utilizada por la División de Sistemas de la Universidad Tecnológica de Pereira.

Queda demostrado que el instrumento de inteligencia artificial es capaz de capturar o materializar la capacidad predictiva, ya que es capaz de comprender las reglas internas que están implícitas en los desarrollos debido a la metodología creada por la División de Sistemas, reglas que nunca fueron descritas y que a pesar de ello las pudo capturar, a diferencia de los expertos en análisis técnico para los cuales estas reglas no son apreciables. Esto solamente muestra un acercamiento a la capacidad de la integración de inteligencia artificial en el trading.

10. RECOMENDACIONES Y TRABAJOS FUTUROS

Una mayor experimentación con nuevos casos de estudio podría ser recomendable para corroborar lo expuesto en la sección anterior.

La interacción entre diferentes ramas de la Inteligencia Artificial podría optimizar el algoritmo de predicción, por ejemplo, la integración de La Red Neuronal con Lógica Difusa o con Algoritmos Genéticos.

Es importante que la red se esté actualizando constantemente con nuevos datos adaptando los pesos de la red para mejorar su capacidad de predicción.

10.1 Trabajos Futuros

MODELO DE PREDICCIÓN DE FALLOS PARA TODO PROYECTO DE SOFTWARE UTILIZANDO REDES NEURONALES: Este proyecto pretende crear un modelo general para todo proyecto de software utilizando Redes Neuronales, cabe resaltar que este proyecto diseñó un modelo específico para el software de la División de Sistemas de la Universidad Tecnológica de Pereira.

MODELO DE PREDICCIÓN DE FALLOS PROYECTOS DE SOFTWARE UTILIZANDO REDES NEURONALES Y LÓGICA DIFUSA: Sería de gran interés experimentar con otros algoritmos diferentes al utilizado (Retropropagación) y con diferentes modelos de red como la red Neurodifusa. Esto permitiría establecer comparaciones entre diferentes tipos de redes y probablemente mejorar las predicciones.

MODELO DE PREDICCIÓN DE FALLOS PARA PROYECTOS DE SOFTWARE EN LA ETAPA DE DISEÑO UTILIZANDO REDES NEURONALES: La gravedad del error y la etapa en que es detectado es de vital importancia en el desarrollo de software. Cuando los errores son detectados en etapas finales como la etapa de prueba la corrección del error es más costosa en términos de tiempo y dinero ya que toda la estructura del software esta creada y una modificación podría afectar el funcionamiento de este creando errores más grandes. Entre más temprano sea detectado el error en sus etapas iniciales será más fácil de corregir, sin embargo, detectar un error en etapas tempranas es difícil ya que no se tiene establecidas variables concretas de medición por ejemplo en la etapa de Diseño. Un proyecto que establezca un modelo de medición de software en su etapa de diseño nos permitirá reducir la tendencia a fallos de un software ya que este sería detectado en una etapa temprana.

11. Bibliografía

A. K. Jain, J. Mao, K. Mohiuddin. «Artificial Neural Networks: A Tutorial.» *IEEE ComputerSpecial Issue on Neural Computing*, 1996.

Abdelmalik Moujahid, Iñaki Inza Pedro Larrañaga. *Algoritmos Gnéticos.* Universidad del País Vasco-Euskal Herriko Unibertsitatea: Departamento de Ciencias de la Computacióon e Inteligencia Artificial, s.f.

Aguirre, Luz Maria Cruz. *Metricas Para El Desarrollo de Software.* 2012.

al., A. Davis et. «Identifying and measuring quality in a software requirements specification Proc. First International Software Metrics Symposium.» 1993. 141-152.

Albrecht, A.J. «Measuring application development.» *Proc. IBM Applications Development Joint SHARE/GUIDE Symposium.* Monterey, 1979. 83-92.

Arias, Bibiana Patricia, y Carlos Cardona. *Algortimos Genéticos y Agentes Inteligentes.* Pereira, 1997.

Berthold, M., y D.J. (eds.) Hand. *Intelligent Data Analysis. An Introduction.* Springer, 2003.

Bertona, Luis Federico. *ENTRENAMIENTO DE REDES NEURONALES BASADO EN ALGORITMOS EVOLUTIVOS.* Buenos Aires, Noviembre 2005.

Bigus, J. *Data Mining with neural networks.* USA: Mc GrawHill, 1996.

Boehm, B.W., B. Clark, y E. Horowitz et al. «Cost models for future life cycle processes: COCOMO 2.0.» *Annals Software Engineering 1(1).* 1995. 1-24.

Boyd, Kaastray. «Designing a neural network for forecasting financial and economic series.» *Neurocomputing* 10 (1996): 215–236.

Briand, L.C., y J.W. Daly y J.K. Wüst. «A unified framework for coupling measurement in objectoriented system.» *IEEE Transaction on Software Engineering*, 1999: 91-121.

Brykczynskki, B. «A survey of software inspection checklist, ACM Software Engineering Notes.» *ACM SIGSOFT*, 1999: 82-89.

Cosín, Javier Tuya Isabel Ramos Román José Javier Dolado. *Técnicas cuantitativas para la gestión en la ingeniería del software.* Netbiblo, 2007.

CRIE. *Universidad Tecnológica de Pereira.* s.f. http://www.utp.edu.co/vicerrectoria/administrativa/division-de-sistemas/noticias/conozca-mas-de-nuestra-division.html.

Deirdre R. Meldrum, Cynthia E. Taylor. *Freeway Traffic Data Prediction Using Artificial Neural Networks and Development of a Fuzzy Logic Ramp Metering Algorithm.* Washington, 1995.

Del Brío, B. M., Sanz Molina, A. *Redes neuronales y Sistemas difusos.* Alfaomega Grupo Editor, 2002.

DeMarco, T. *Controlling software projects.* Yourdon Press, 1982.

Dorffner, Georg. *Neural Networks for Time Series Processing.* Department of medical ciberneting and Artificial Intelligence: University of Viena and Austrian Research Institute for Artificial Intelligence, 2008.

Duarte, Joanna Verónica Collantes. *Predicción con Redes Neuronales: Comparación con las metodologías de Box y Jenkis.* Mérida, Venezuela, 2001.

Ernesto González Díaz, Zady Pérez Hernández,Ivet Espinosa Conde. «Técnicas de minería de datos .» s.f. http://www.monografias.com/trabajos55/mineria-de-datos/mineria-de-datos2.shtml#refer.

Farbey, B. «Software Quality metrics: considerations about requirements and requirements specification.» *Information and Software Technology*, 1990: 60-64.

Fayyad, U. M., Piatetsky-Shapiro, G., Smith, P., Uthurusamy R. *Advances in Knowledge Discovery and Data-Mining.* AAAI Press / The MIT Press, 1996.

Fleur W. Op 't Landt, F. W. Op 't L. *Stock Price Prediction using Neural Networks.* 1997.

Frank, Ian H. Witten and Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 2000.

French, J.C., y J.C. Knight y A.L. Powell. «Applying hipertext structures to software documentation.» *Information Processing and Management*, 1997: 219-231.

GARZON, DIEGO FERNANDO SOCHA, y GILBERTO ANTONIO ORTIZ HERRADA. *APLICACIÓN DE REDES NEURONALES MLP A LA PREDICCION DE UN PASO EN SERIES DE TIEMPO.* Bogotá, 2006.

Genero, M., M.E. Manso, y M. Piattini y F.J. García. «Assessing the quality and the complexity of OMT models.» *Proc. 2nd European Software Measurements Conference-FESMA 99.* Amsterdam, Netherlands, 1999. 99-109.

Genero, M., y y C. Calero M. Piattini. «Una propuesta para medir la calidad de los diagramas de clases en UML.» *2000.* Cancun, México, 2000. 373-384.

González, Daniel Santín. *EFICIENCIA TÉCNICA Y REDES NEURONALES: UN MODELO PARA EL CÁLCULO DEL VALOR AÑADIDO EN EDUCACIÓN.* Madrid, 2003.

Gordillo, Lic. Silvia. «MODELIZACIÓN DE CAMPOS CONTINUOS EN SISTEMAS DE. INFORMACIÓN GEOGRÁFICA.» 1998.

Gurney, K. *Neural Nets.* CRC Press, 1997.

Gutiérrez, Hugo Flores. *Redes Neuronales Artificiales 1.* Instituto Politecnico Nacional. s.f. http://www.hugo-inc.com/RNA/Unidad%201/1.5.html.

Hansen, A, y J Mouritsen. *Managerial technologies and netted and Netted Networks: Competitiveness in Action - The work of translating perfomance in a high tech firm, Organization.* 1999.

Isasi Viñuela, P., Galván León. *Redes Neuronales Artificiales. Un enfoque Práctico.* enfoque Práctico, 2004.

J. L. Sang, K. Siau. *A Review of data mining techniques.* MCB University Press, 2001.

Kemerer, S.R. Chidamber y C.F. «A Metrics Suite for Object-Oriented Design.» *IEEE Transactions of Software Engineering*, 1994: 476-493.

L, Fausett. *Fundamentals of Neural Networks.* Englewood : Prentice Hall, 1992.

Lehner, F. «Quality control in software documentation: Measurement of text comprehensibility.» *Information and Management*, 1993: 133-146.

Levene, J. Borges and M. «Data mining of User Navigation Paterns.» *Proceedings WEBKDD´99*, 1999: 92-111.

Mannila, H. *Data Mining: Machine Learning, Statistics and Databases.* Dep. of Computer: University of Helsinky, s.f.

Martinez, José H. Hilera/Victor J. *Redes Neuronales Artificiales. Fundamentos Modelos y Aplicaciones.* Mexico: Alfaomega, 2000.

Mayrhauser, Anneliese Von. «Testing and evolutionary development.» *ACM DL* 16 (1991).

Microsoft. *Conceptos de minería de datos.* s.f. https://msdn.microsoft.com/es-es/library/ms174949%28v=sql.120%29.aspx.

Moreno, M.N., F.J. García, M.J. Polo, y V. López y A. González. *Marco de referencia para la gestión de la calidad de las especificaciones de requisitos.* Lisboa, Portugal: Proc. QUATIC.2001, 2001.

Nascimento. *Artificial Neural Networks in Control and Optimization.* Manchester, United Kingdom: University of Manchester, Institute of Science and Technology (UMIST), 1994.

Olabe, Xabier Basogain. «REDES NEURONALES ARTIFICIALES Y SUS APLICACIONES.» s.f. http://cvb.ehu.es/open_course_ware/castellano/tecnicas/redes_neuro/contenidos/pdf/libro-del-curso.pdf.

Orallo Hernández, J.:Quintana Ramírez, Ma. J..:Ramírez Ferri. *Introducción a la Minería de Datos.* Prentice Hall, 2004.

Park, Lee H. Chong and Kyoung Cheol. «Prediction of montly transition of the composition stock price index using recurrent back-propagation.» *Artificial Neural Networks.* 2 Elsevier Science Publishers B.V. 1992, 1992. 1629-1632.

Poels, G. «On the measurements of event-based object-oriented conceptual models.» *Proc. 4th International ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering.* Cannes, France, 2000.

—. «Towards a size measurement framework for object-oriented especifications.» *Proc. 1st European Software Measurement Conference - FESMA'98.* Antwerp, 1998. 379-388.

—. «Towards a size measurement framework for object-oriented especifications.» *Proc. 1st European Software Measurement Conference - FESMA'98.* Antwerp, 1998. 379-388.

Roth, T., y P. Aiken y S. Hobbs. «Hypermedia support for software developmemt: a retrospective assessment.» (Hypermedia) 1994: 149-173.

S., Minsky M & Papert. *Perceptrons: an introduction to computacional geometry.* Cambridge: The MIT Press, 1969.

Saavedra, Carlos, y Fernando Izaurieta. *Redes Neuronales Artificiales.* Chile: Departamento de Física, s.f.

Samson, W.B., y D.G. Nevill y P.I. Dugard. «Predictive software metrics based on a formal specification.» *Software Engineering Journal*, 1990: 5(1).

Sanz, Bonifacio Martín y Alfredo. *Redes Neuronales y Sistemas Difusos.* Alfaomega-Rama, 2002.

Shepperd, N.I. Churcher y M.J. «Towards Conceptual Framework for Object-Oriented Metrics.» *ACM Software Engineering Notes*, 1995: 67-76.

Skapura, J.A. Freeman y D.M. *Redes Neuronales: Algoritmos, Aplicaciones y Técnicas de Programación.* Addison-Wesley, 1993.

Srikant, Ramakrishana. *Research Report: Mining Sequential Patters: Generalizations and performance improvements.* Avignon, France: EDBT, 1998.

Stevens, J.D. Arthur y K.T. «Assessing the adequacy of documentation through document quality indicators.» *Proc. the IEEE Conference of Software Maintenance*, 1989: 40-49.

TIC, Centro de investigación. «Inteligencia Artificial y Aplicaciones.» s.f. http://citic-research.org/area_tecnologica/2?locale=es.

Villena, J. «Apuntes y documentación de Inteligencia en redes de Comunicaciones (5º Curso Ing. de Telecomunicación).» s.f. http://www.it.uc3m.es/jvillena/irc/indice.html.

Widrow, B. «Adaptative sampled-data systems, a statistical theory of adaptation.» *IRE WESCON Convention Record.* New York: Institure of Radio Engineers, 1959.

X, Yao. «Evolving Artificial Neural Networks. .» *Proceedings of the IEEE* 87(9) (1999): 1423-1447.

Yaidelyn Macías Rivero, María Victoria Guzmán Sánchez,Yamila Martínez Suárez. «Modelo de evaluación para software que emplean indicadores métricos en la vigilancia científico-tecnológica.» Diciembre de 2009. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352009001200003.

ANEXOS

Anexo A Código fuente Red Neuronal

```
unit entrenar;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, jpeg, Buttons, ComCtrls;

type
  Tfraentrenar = class(TFrame)
    GroupBox1: TGroupBox;
    cmbentrada: TComboBox;
    Label1: TLabel;
    cmboculta: TComboBox;
    cmbsalida: TComboBox;
    Label2: TLabel;
    Label3: TLabel;
    pnlControl: TPanel;
    grpbred: TGroupBox;
    Image1: TImage;
    Ant: TSpeedButton;
    Sig: TSpeedButton;
    controlador: TTimer;
    Salir: TSpeedButton;
    Label4: TLabel;
    edterror: TEdit;
    Iniciar: TSpeedButton;
```

```
    Detener: TSpeedButton;
    Guardar: TSpeedButton;
    nomred: TEdit;
    Label5: TLabel;
    Label6: TLabel;
    edtnum_reg: TEdit;
    Image2: TImage;
    Label7: TLabel;
    ecmo: TLabel;
    TrackBar1: TTrackBar;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    procedure cmbentradaChange(Sender: TObject);
    procedure cmbocultaChange(Sender: TObject);
    procedure cmbsalidaChange(Sender: TObject);
    procedure iniciarPesos;
    procedure iniciarClick(Sender: TObject);
    procedure KeyPress(Sender: TObject; var Key: Char);
    procedure SigClick(Sender: TObject);
    procedure cargar_edt_a_matriz;
    procedure cargar_matriz_a_edt;
    procedure AntClick(Sender: TObject);
    procedure detenerClick(Sender: TObject);
    procedure controladorTimer(Sender: TObject);
    procedure calcularnet_h_j;
    procedure calcularnet_o_k;
    procedure calcular_delta_k;
    procedure calcular_delta_j;
```

```
    procedure actualizarPesos;
    procedure escalamiento;
    procedure GuardarClick(Sender: TObject);
    procedure edtnum_regKeyPress(Sender: TObject; var Key: Char);
    procedure edterrorKeyPress(Sender: TObject; var Key: Char);
    procedure edtnum_regExit(Sender: TObject);
    procedure SalirClick(Sender: TObject);
    procedure cargar_matriz_a_edt_vis;
    procedure TrackBar1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  type
    patron = record
    x: Array[0..20] of double;
    y_j: Array[0..20] of double;
    y_k: Array[0..20] of double;
    d: Array[0..20] of double;
    net_h_j:Array[0..20] of double;
    net_o_k:Array[0..20] of double;
    delta_h_j:Array[0..20] of double;
    delta_o_k:Array[0..20] of double;
  end;

var
    Edtentrada:Array[0..20] Of TEdit;
    Edtsalida:Array[0..20] Of TEdit;
```

```
    Edtoculta:Array[0..20] Of TEdit;
    Lblentrada:Array[0..20] of TLabel;
    Lbloculta:Array[0..20] of TLabel;
    Lblsalida:Array[0..20] of TLabel;
    M,L,N:Integer;
    W_j_i:Array[0..20,0..20] of double;
    W_k_j:Array[0..20,0..20] of double;
    Momento_W_j_i:Array[0..20,0..20] of double;
    Momento_W_k_j:Array[0..20,0..20] of double;
    Patrones:Array[0..200] of patron;
    aux_delta_o_k: Double;
    p:integer;
    auxp:integer;
    max:integer;
    min:integer;
    num_reg:integer;
    epocas:integer;
    error:Double;

implementation

uses StrUtils;

{$R *.dfm}

procedure Tfraentrenar.cmbentradaChange(Sender: TObject);
var i, iTop:Integer;
begin
num_reg:=10;
 iTop:=30;
```

```
    {Crea los TEdit...}
for i:=0 to N+1 do
  begin
    if (Edtentrada[i] is TEdit)then
       Edtentrada[i].free;
       Lblentrada[i].Free;
  end;

    For i:=Low(Edtentrada) to cmbentrada.ItemIndex+1 Do
    Begin
        Edtentrada[i]:=TEdit.Create(Self);



        Edtentrada[i].Parent:=Self;
        Edtentrada[i].Left :=300;
        Edtentrada[i].Top  :=iTop;
        Edtentrada[i].Width:=30;
        Edtentrada[i].OnKeyPress:=KeyPress;

        Lblentrada[i]:=TLabel.Create(Self);
        Lblentrada[i].Parent := Self.grpbred ;
        Lblentrada[i].Left := 70;
        Lblentrada[i].Top :=  iTop;
        Lblentrada[i].Caption:='N_e_'+IntToStr(i);

        if i=0 then begin
          Edtentrada[i].Color:=clBlack;
          Edtentrada[i].Enabled:=False;
          Edtentrada[i].Text:='1';
        end else begin
```

```
            Edtentrada[i].Color:=clYellow;
            Edtentrada[i].Text:='0';
          end;
          Inc(iTop, Edtentrada[i].Height+2);

     End;
     N:=cmbentrada.ItemIndex;
     cmboculta.Enabled:=true;
end;

procedure Tfraentrenar.cmbocultaChange(Sender: TObject);
var i, iTop:Integer;
begin
 iTop:=50;

for i:=0 to L+1 do
  begin
    if (Edtoculta[i] is TEdit)then
       Edtoculta[i].free;
       Lbloculta[i].Free;
  end;
     For i:=Low(Edtoculta) to cmboculta.ItemIndex+1 Do
     Begin
          Edtoculta[i]:=TEdit.Create(Self);
          Edtoculta[i].Parent:=Self;
          Edtoculta[i].Left :=500;
          Edtoculta[i].Top  :=iTop;
          Edtoculta[i].Width:=30;
          Edtoculta[i].Enabled:=False;
```

```
        Lbloculta[i]:=TLabel.Create(Self);
        Lbloculta[i].Parent := Self.grpbred ;
        Lbloculta[i].Left := 270;
        Lbloculta[i].Top :=  iTop;
        Lbloculta[i].Caption:='N_o_'+IntToStr(i);

        if i=0 then begin
          Edtoculta[i].Color:=clBlack;
          Edtoculta[i].Text:='1';
        end else begin
          Edtoculta[i].Color:=clRed;
        end;
        Inc(iTop, Edtoculta[i].Height+2);
    End;
    L:=cmboculta.ItemIndex;
    cmbsalida.Enabled:=true;
end;

procedure Tfraentrenar.cmbsalidaChange(Sender: TObject);
var i, iTop:Integer;
begin
for i:=0 to M+1 do
  begin
   if (Edtsalida[i] is TEdit)then
      Edtsalida[i].free;
      Lblsalida[i].Free;
  end;

 iTop:=70;
    For i:=Low(Edtsalida) to cmbsalida.ItemIndex+1 Do
```

```
    Begin
        Edtsalida[i]:=TEdit.Create(Self);
        Edtsalida[i].Parent:=Self;
        Edtsalida[i].Left :=700;
        Edtsalida[i].Top  :=iTop;
        Edtsalida[i].Width:=30;

        Lbloculta[i]:=TLabel.Create(Self);
        Lbloculta[i].Parent := Self.grpbred ;
        Lbloculta[i].Left := 470;
        Lbloculta[i].Top :=  iTop;
        Lbloculta[i].Caption:='N_s_'+IntToStr(i);

        Edtsalida[i].OnKeyPress:=KeyPress;
        if i=0 then begin
          Edtsalida[i].Color:=clBlack;
          Edtsalida[i].Enabled:=False;
          Edtsalida[i].Text:='1';
        end else begin
          Edtsalida[i].Color:=clWhite;
          Edtsalida[i].Text:='0';
        end;
//          Edtentrada[i].OnKeyPress:=va
        Inc(iTop, Edtsalida[i].Height+2);
    End;
    M:=cmbsalida.ItemIndex;
    grpbred.Enabled:=true;
//   edterror.Enabled:=true;
    edtnum_reg.Enabled:=true;
    pnlControl.Enabled:=true;
```

```
    p:=1;
end;

procedure Tfraentrenar.iniciarPesos;
var i,j,k:Integer;
begin
Randomize;
For j:=Low(Edtoculta) to L+1 Do
Begin
  For i:=Low(Edtentrada) to N+1 Do
    Begin
      W_j_i[j,i]:=Random;
    End;
End;
Randomize;
For k:=Low(Edtsalida) to M+1 Do
Begin
   For j:=Low(Edtoculta) to L+1 Do
      Begin
        W_k_j[k,j]:=Random;
      End;
   End;
end;

procedure Tfraentrenar.iniciarClick(Sender: TObject);
begin
cargar_edt_a_matriz;
//escalamiento;
iniciarPesos;
grpbred.Enabled:=false;
```

```
epocas:=1;
error:=StrToFloat(edterror.text);
controlador.Enabled:=true;
Detener.Enabled:=true;
end;



procedure Tfraentrenar.KeyPress(Sender: TObject; var Key: Char);
begin
     if not (key in ['0'..'9',',',#8]) then
   begin
     key:=#0;
   end
end;

procedure Tfraentrenar.SigClick(Sender: TObject);
begin
if p<num_reg then begin
  if Patrones[p].x[0]=1 then begin
    p:=p+1;
    cargar_matriz_a_edt;
  end else begin
    cargar_edt_a_matriz;
    p:=p+1;
  end;

  grpbred.Caption:='Red '+IntToStr(p);
  Ant.Enabled:=true;
//  if p=20 then begin
//    Sig.Enabled:=true;
```

```
//  end;
  iniciar.Enabled:=true;
end;
cmbentrada.Enabled:=false;
cmbsalida.Enabled:=false;
cmboculta.Enabled:=false;
Iniciar.Enabled:=true;
end;

procedure Tfraentrenar.escalamiento;
var i,r,index:integer;
var valor:double;
begin

max:=0;
For r:=1 to p Do Begin
  For i:=Low(Edtentrada)+1 to N+1 Do
    Begin
      if Trunc(Patrones[r].x[i])>max then begin
        max:=Trunc(Patrones[r].x[i]);
      end;
    End;
  index:=Low(Edtentrada)+1;
  min:=Trunc(Patrones[r].x[index]);
  For i:=Low(Edtentrada)+1 to N+1 Do
    Begin
      if Patrones[r].x[i]<min then begin
        min:=Trunc(Patrones[r].x[i]);
      end;
    End;
```

```
  For i:=Low(Edtentrada)+1 to N+1 Do
    Begin
   if i=Low(Edtentrada) then begin
      valor:=Patrones[r].x[i];
   end else begin
   if (max-min)<>0 then begin
      valor:=(Patrones[r].x[i]-min)/(max-min);
   end else begin
      valor:=StrToFloat('1');
   end;
   end;
   Patrones[r].x[i]:=valor;
  End;

  End;
end;

procedure Tfraentrenar.cargar_edt_a_matriz;
var i:integer;
begin

For i:=Low(Edtentrada) to cmbentrada.ItemIndex+1 Do
  Begin
     Patrones[p].x[i]:=StrToFloat(Edtentrada[i].Text);
     if i=0 then begin
       Edtentrada[i].Text:='1';
     end else begin
       Edtentrada[i].Text:='0';
     end;
```

```
  End;
For i:=Low(Edtsalida) to cmbsalida.ItemIndex+1 Do
  Begin
     Patrones[p].d[i]:=StrToFloat(Edtsalida[i].Text);
     if i=0 then begin
       Edtsalida[i].Text:='1';
     end else begin
       Edtsalida[i].Text:='0';
     end;

  End;

end;

procedure Tfraentrenar.cargar_matriz_a_edt;
var i:integer;
begin
For i:=Low(Edtentrada) to cmbentrada.ItemIndex+1 Do
  Begin
     Edtentrada[i].Text:=FloatToStr(Patrones[p].x[i]);
  End;
For i:=Low(Edtsalida) to cmbsalida.ItemIndex+1 Do
  Begin
     Edtsalida[i].Text:=FloatToStr(Patrones[p].d[i]);
  End;

end;

procedure Tfraentrenar.AntClick(Sender: TObject);
```

```
begin
if p>1 then begin
  cargar_edt_a_matriz;
  p:=p-1;
  cargar_matriz_a_edt;
  grpbred.Caption:='Red '+IntToStr(p);
  if p=1 then begin
    Ant.Enabled:=true;
  end;
 end;
end;

procedure Tfraentrenar.detenerClick(Sender: TObject);
begin
controlador.Enabled:=false;
Guardar.Enabled:=true;
end;

procedure Tfraentrenar.controladorTimer(Sender: TObject);
var opcion:integer;
var valorf:Double;
begin
if epocas<2000 then begin
  cargar_matriz_a_edt_vis;
  calcularnet_h_j;
  calcular_delta_j;
  Randomize;
  valorf:=Random;
  ecmo.Caption:=FloatToStr(valorf);
  if valorf<0.01 then begin
```

```
    controlador.Enabled:=false;
  end;
  auxp:=auxp+1;
  if auxp=num_reg+1 then begin
    auxp:=1;
  end;
  epocas:=epocas+1;
end else begin
  controlador.Enabled:=false;
end;
end;

procedure Tfraentrenar.cargar_matriz_a_edt_vis;
var i:integer;
begin
For i:=Low(Edtentrada) to cmbentrada.ItemIndex+1 Do
  Begin
    Edtentrada[i].Text:=FloatToStr(Patrones[auxp].x[i]);
  End;
For i:=Low(Edtsalida) to cmbsalida.ItemIndex+1 Do
  Begin
    Edtsalida[i].Text:=FloatToStr(Patrones[auxp].d[i]);
  End;

end;



procedure Tfraentrenar.calcularnet_h_j;
var i,j:integer;
begin
```

```
for j:=1 to L do begin
  for i:=1 to N do begin
    Patrones[auxp].net_h_j[j]:=Patrones[auxp].net_h_j[j]+(W_j_i[j][i]*Patrones[p].x[i]);
  end;
  Patrones[auxp].y_j[j]:=1/(1+Exp(Patrones[auxp].net_h_j[j]*-1));
end;
end;

procedure Tfraentrenar.calcularnet_o_k;
var k,j:integer;
begin

for k:=1 to M do begin
  for j:=0 to L do begin

Patrones[auxp].net_o_k[k]:=Patrones[auxp].net_o_k[k]+(W_k_j[k][j]*Patrones[p].y_j
[j]);
  end;
  Patrones[auxp].y_k[k]:=1/(1+Exp(Patrones[auxp].net_o_k[k]*-1));
end;
end;

procedure Tfraentrenar.calcular_delta_k;
var k:integer;
begin
for k:=0 to M do begin
  Patrones[auxp].delta_o_k[k]:=(Patrones[auxp].d[k]-
Patrones[auxp].y_k[k])*Patrones[auxp].y_k[k]*(1-Patrones[auxp].y_k[k]);
end;
```

```
end;

procedure Tfraentrenar.calcular_delta_j;
var j,k:integer;
begin
for j:=0 to L do begin
  aux_delta_o_k:=0.0;
  for k:=0 to M do begin
      aux_delta_o_k:=aux_delta_o_k+(Patrones[auxp].delta_o_k[k]*W_k_j[k][j]);
  end;
  Patrones[auxp].delta_h_j[j]:=Patrones[auxp].y_j[j]*aux_delta_o_k;
end;
end;

procedure Tfraentrenar.actualizarPesos;
var i,j,k:Integer;
begin
     For j:=Low(Edtoculta) to L+1 Do
     Begin
          For i:=Low(Edtentrada) to N+1 Do
          Begin
            W_j_i[j,i]:=W_j_i[j,i]+Patrones[auxp].delta_h_j[j]*Patrones[auxp].x[i];
          End;
     End;

  For k:=Low(Edtsalida) to M+1 Do
  Begin
      For j:=Low(Edtoculta) to L+1 Do
        Begin
          W_k_j[k,j]:=W_k_j[k,j]+Patrones[auxp].delta_o_k[k]*Patrones[auxp].y_j[i];
```

```
        End;
   End;
End;



procedure Tfraentrenar.GuardarClick(Sender: TObject);
var
  F: TextFile;
  j,k,i:integer;
  Linea:String;
begin
if nomred.Text<>'' then begin
  AssignFile( F, ExtractFilePath( Application.ExeName ) + nomred.Text+'.txt' );
  Rewrite( F );
  Linea:=IntToStr(N)+';'+IntToStr(L)+';'+IntToStr(M)+';'+ecmo.Caption+';';
  Writeln(F,Linea);
  Linea:='';
  For j:=Low(Edtoculta) to L+1 Do
  Begin
   For i:=Low(Edtentrada) to N+1 Do
     Begin
       Linea:=Linea+FormatFloat('0.0000',W_j_i[j,i])+';';
     End;
   WriteLn(F,Linea);
  End;
  Linea:='';
  For k:=Low(Edtsalida) to M+1 Do
  Begin
    For j:=Low(Edtoculta) to L+1 Do
      Begin
```

```
        Linea:=Linea+FormatFloat('0.0000',W_k_j[k,j])+';';
      End;
      WriteLn(F,Linea);
  End;

  WriteLn( F, 'Esto es el contenido del archivo de texto.' );
  CloseFile( F );
  ShowMessage('Archivo Creado');
end else begin
  ShowMessage('Debe escribir un nombre de archivo!');
end;
end;
procedure Tfraentrenar.edtnum_regKeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',#8]) then
  begin
    key:=#0;
end

end;

procedure Tfraentrenar.edterrorKeyPress(Sender: TObject; var Key: Char);
begin
if not (key in ['0'..'9',',',#8]) then
  begin
    key:=#0;
end
end;

procedure Tfraentrenar.edtnum_regExit(Sender: TObject);
```

```
begin
num_reg:=StrToInt(edtnum_reg.Text);
end;

procedure Tfraentrenar.SalirClick(Sender: TObject);
begin
cmbentrada.Enabled:=true;
cmbentrada.ItemIndex:=-1;
cmboculta.ItemIndex:= -1;
cmboculta.Enabled:=false;
cmbsalida.ItemIndex:= -1;
cmbsalida.Enabled:=false;
edterror.Text:='0';
edterror.Enabled:=false;
edtnum_reg.Text:='0';
edtnum_reg.Enabled:=False;
nomred.Text:='';
Iniciar.Enabled:=false;
Guardar.Enabled:=false;
Detener.Enabled:=false;
end;

procedure Tfraentrenar.TrackBar1Change(Sender: TObject);
begin
controlador.Interval:=controlador.Interval-100;
end;

end.
unit evaluar;
```

```
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls, Buttons;

type
  TPredecir = class(TFrame)
    GroupBox1: TGroupBox;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    predecir: TSpeedButton;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    grpbred: TGroupBox;
    Label1: TLabel;
    ecm: TLabel;
    SpeedButton3: TSpeedButton;
    Image1: TImage;
    Label2: TLabel;
    Label3: TLabel;
    procedure SpeedButton1Click(Sender: TObject);
    procedure crearentrada;
    procedure KeyPress(Sender: TObject; var Key: Char);
    procedure crearoculta;
    procedure crearsalida;
    Function Split(enumeracion : string): TStringList;
    procedure escalamiento;
    procedure predecirClick(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  type
    patron = record
    x: Array[0..20] of double;
    y_j: Array[0..20] of double;
    y_k: Array[0..20] of double;
    d: Array[0..20] of double;
    net_h_j:Array[0..20] of double;
    net_o_k:Array[0..20] of double;
    delta_h_j:Array[0..20] of double;
    delta_o_k:Array[0..20] of double;
  end;

  var
  M,L,N,max,min:Integer;
  Edtentrada:Array[0..20] Of TEdit;
  Edtsalida:Array[0..20] Of TEdit;
  Edtoculta:Array[0..20] Of TEdit;
  Lblentrada:Array[0..20] of TLabel;
  Lbloculta:Array[0..20] of TLabel;
  Lblsalida:Array[0..20] of TLabel;
  Patrones:patron;


implementation
```

```
uses StrUtils;

{$R *.dfm}

Function TPredecir.Split(enumeracion : string): TStringList;
var
	Aux : String;
	i : integer;
	Lista :TStringList;
begin
	enumeracion := trim(enumeracion);
	Lista := TStringList.create;
	Aux := '';
	i:=1;
	while i <= length(Enumeracion) do begin
		if enumeracion[i] <> ';' then begin
			Aux := Aux + enumeracion[i];
		end;
		if enumeracion[i] = ';' then begin
			Lista.Add(trim(Aux));
			Aux := '';
		end;
		inc(i);
	end;
	if Aux <> '' then begin
	    Lista.Add(trim(Aux));
	end;
	result := lista;
end;
```

```
procedure TPredecir.SpeedButton1Click(Sender: TObject);
var F: TextFile;
    sLinea: String;
    openDialog : TOpenDialog;
    encabezado: integer;
    OutPutList: TStringList;
begin
  openDialog := TOpenDialog.Create(self);
  openDialog.InitialDir := GetCurrentDir;
  openDialog.Options := [ofFileMustExist];
  openDialog.Filter := 'Archivos de la Red|*.txt';
  openDialog.FilterIndex := 2;
  if openDialog.Execute then begin
  ShowMessage('Archivo : '+openDialog.FileName);
  AssignFile( F,openDialog.FileName );
  grpbred.Caption:=openDialog.FileName;
  Reset( F );
  encabezado:=0;
  while not Eof( F ) do
  begin
    ReadLn( F, sLinea );
  if encabezado=0 then begin
      OutPutList:=Split(sLinea);
      N:=StrToInt(OutPutList[0]);
      L:=StrToInt(OutPutList[1]);
      M:=StrToInt(OutPutList[2]);
      ecm.Caption:=OutPutList[3];
      crearentrada;
      crearoculta;
```

```
      crearsalida;
      encabezado:=1;
    end;
  end;
 CloseFile( F );
   end else begin ShowMessage('Operación Cancelada');end;

end;

procedure TPredecir.crearoculta;
var i, iTop:Integer;
begin
 iTop:=50;

     For i:=low(Edtoculta) to L Do
     Begin
          Edtoculta[i]:=TEdit.Create(Self);
          Edtoculta[i].Parent:=Self.grpbred;
          Edtoculta[i].Left :=300;
          Edtoculta[i].Top  :=iTop;
          Edtoculta[i].Width:=30;
          Edtoculta[i].Enabled:=False;

          Lbloculta[i]:=TLabel.Create(Self);
          Lbloculta[i].Parent := Self.grpbred;
          Lbloculta[i].Left := 255;
          Lbloculta[i].Top :=  iTop+5;
          Lbloculta[i].Caption:='N_o_'+IntToStr(i);

          if i=0 then begin
```

```
            Edtoculta[i].Color:=clBlack;
            Edtoculta[i].Text:='1';
          end else begin
            Edtoculta[i].Color:=clRed;
          end;
          Inc(iTop, Edtoculta[i].Height+2);
      End;
end;

procedure TPredecir.crearentrada;
var i, iTop:Integer;
begin
iTop:=30;
      For i:=Low(Edtentrada) to N Do
      Begin
          Edtentrada[i]:=TEdit.Create(Self);
          Edtentrada[i].Parent:=Self.grpbred;
          Edtentrada[i].Left :=100;
          Edtentrada[i].Top  :=iTop;
          Edtentrada[i].Width:=30;
          Edtentrada[i].OnKeyPress:=KeyPress;

          Lblentrada[i]:=TLabel.Create(Self);
          Lblentrada[i].Parent := Self.grpbred ;
          Lblentrada[i].Left := 55;
          Lblentrada[i].Top :=  iTop+5;
          Lblentrada[i].Caption:='N_e_'+IntToStr(i);

          if i=0 then begin
            Edtentrada[i].Color:=clBlack;
```

```
            Edtentrada[i].Enabled:=False;
            Edtentrada[i].Text:='1';
          end else begin
            Edtentrada[i].Color:=clYellow;
            Edtentrada[i].Text:='0';
          end;
          Inc(iTop, Edtentrada[i].Height+2);
      End;
end;

procedure TPredecir.crearsalida;
var i, iTop:Integer;
begin
 iTop:=70;
      For i:=Low(Edtsalida) to M Do
      Begin
          Edtsalida[i]:=TEdit.Create(Self);
          Edtsalida[i].Parent:=Self.grpbred;
          Edtsalida[i].Left :=500;
          Edtsalida[i].Top  :=iTop;
          Edtsalida[i].Width:=30;
          Edtsalida[i].Enabled:=false;

          Lblsalida[i]:=TLabel.Create(Self);
          Lblsalida[i].Parent := Self.grpbred ;
          Lblsalida[i].Left := 455;
          Lblsalida[i].Top :=  iTop+5;
          Lblsalida[i].Caption:='N_s_'+IntToStr(i);

          Edtsalida[i].OnKeyPress:=KeyPress;
```

```
        if i=0 then begin
          Edtsalida[i].Color:=clBlack;
          Edtsalida[i].Enabled:=False;
          Edtsalida[i].Text:='1';
        end else begin
          Edtsalida[i].Color:=clWhite;
          Edtsalida[i].Text:='0';
        end;
        Inc(iTop, Edtsalida[i].Height+2);
    End;
    grpbred.Enabled:=true;
end;


procedure TPredecir.KeyPress(Sender: TObject; var Key: Char);
begin
   if not (key in ['0'..'9',',',#8]) then
   begin
     key:=#0;
   end
end;

procedure TPredecir.escalamiento;
var i,index:integer;
var valor:double;
begin

max:=0;
  For i:=1 to N Do
   Begin
```

```
      if Trunc(StrToFloat(Edtentrada[i].text))>max then begin
        max:=Trunc(StrToFloat(Edtentrada[i].text));
      end;
   End;
  index:=1;
  min:=Trunc(StrToFloat(Edtentrada[index].text));
  For i:=1 to N Do
   Begin
     if Trunc(StrToFloat(Edtentrada[i].text))<min then begin
       min:=Trunc(StrToFloat(Edtentrada[i].text));
     end;
   End;
  For i:=0 to N Do
   Begin
  if i=0 then begin
     valor:=Trunc(StrToFloat(Edtentrada[i].text));
   end else begin
     if (max-min)<>0 then begin
       valor:=(StrToFloat(Edtentrada[i].text)-min)/(max-min);
     end else begin
       valor:=StrToFloat('1');
     end;
   end;
   Patrones.x[i]:=StrToFloat(FormatFloat('0.0',valor));
  End;
end;



procedure TPredecir.predecirClick(Sender: TObject);
```

```
var i,contador:integer;
var valor:double;
var avalor:Array[0..9] of String;
begin
contador:=0;
for i:=1 to N Do Begin
  if StrToInt(Edtentrada[i].Text) <>0 then begin
    contador:=1;
  end;
End;
if contador=1 then begin
escalamiento;
if contador=0 then begin
  for i:=1 to M Do Begin
    Randomize;
    valor:=Random;
    Edtsalida[i].Text:=formatfloat('#',valor);
    if Edtsalida[i].Text='' then begin
        Edtsalida[i].Text:='0';
    end;
  End;
end;
end else begin
   ShowMessage('Debe ingresar valores');
end;

end;

procedure TPredecir.SpeedButton2Click(Sender: TObject);
var i:integer;
```

```
begin
ecm.Caption:='0.0';
grpbred.Caption:='';
for i:=0 to N+1 do
  begin
    if (Edtentrada[i]<> NIL)then begin
       Edtentrada[i].free;
   end;
  end;
for i:=0 to L+1 do
  begin
    if (Edtoculta[i]<> NIL)then begin
       Edtoculta[i].free;
    end;
  end;
for i:=0 to M+1 do
  begin
    if (Edtsalida[i]<> NIL)then begin
       Edtsalida[i].free;
    end;
  end;

  for i:=0 to N+1 do
  begin
    if (Lblentrada[i]<> NIL)then begin
       Lblentrada[i].Free;
   end;
  end;
for i:=0 to L+1 do
  begin
```

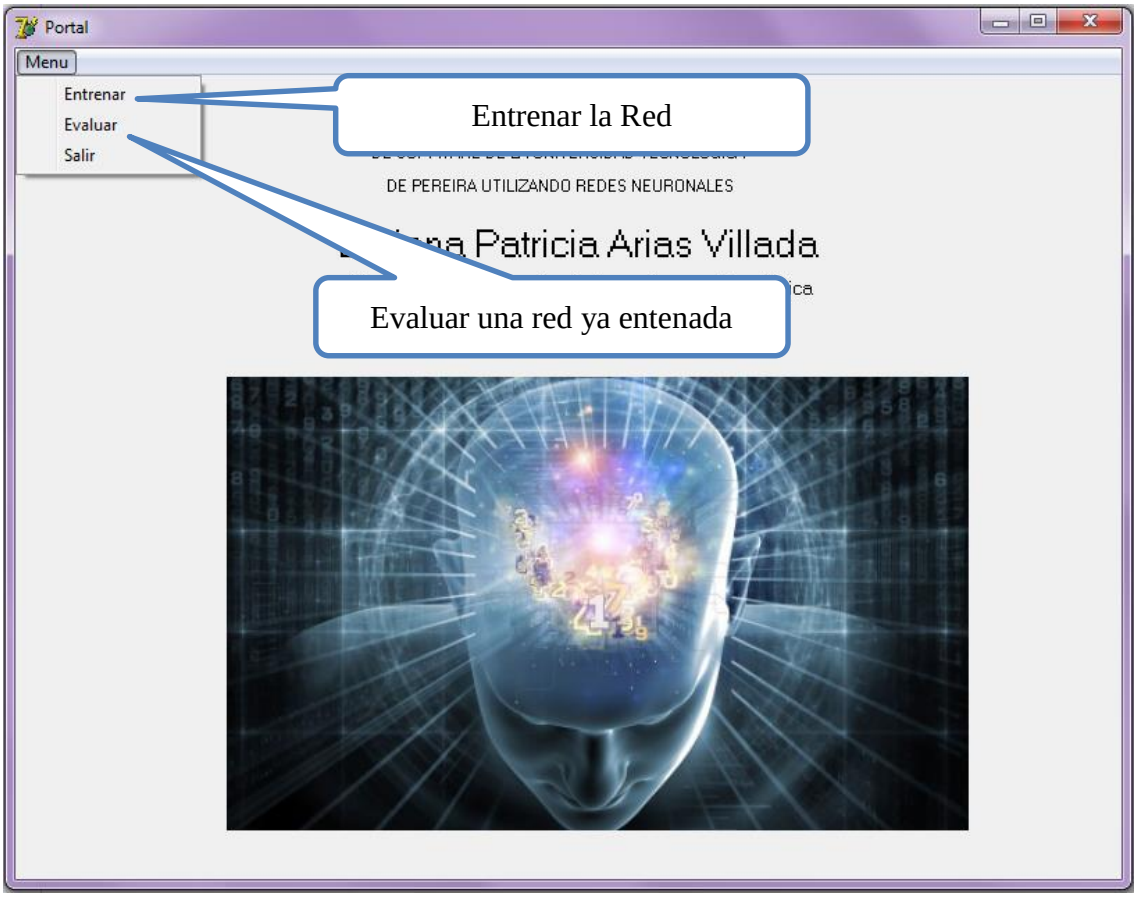
```
    if (Lbloculta[i]<> NIL)then begin
       Lbloculta[i].Free;
    end;
  end;
for i:=0 to M+1 do
  begin
    if (Lblsalida[i]<> NIL)then begin
       Lblsalida[i].free;
    end;
  end;

end;

end.
```

Anexo B Manual de Usuario del Software

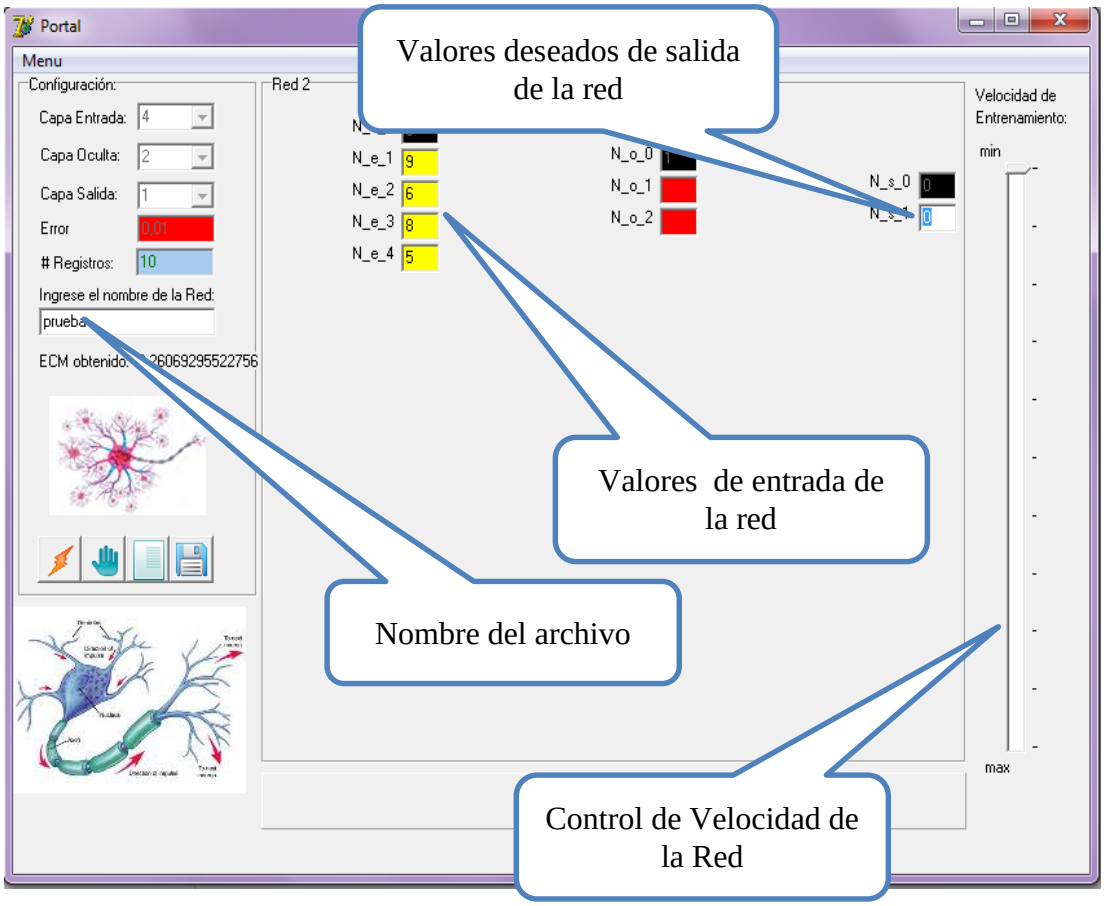
1. *Módulos:*



Entrenar: Módulo encargado de entrenar la red

Evaluar: Después de entrenada la red podemos crear un archivo y luego evaluar y predecir la calidad de dicha red entrenada.

3. *Módulo de entrenamiento:*



Nota: El software puede ser utilizado para entrenar cualquier red de tres capas (entrada, oculta y salida)

Capa de Entrada: # de Neuronas de Entrada

Capa Oculta: # de Neuronas Ocultas

Capa de Salida: Número de Neuronas de Salida.

 # Registros: # de registros de entrenamiento.

Error: ECM(por defecto es de 0.01).

ECM obtenido: Error calculado durante el entrenamiento.

Botones:

 *Entrenar la red*

 *Detener el entrenamiento de la red*

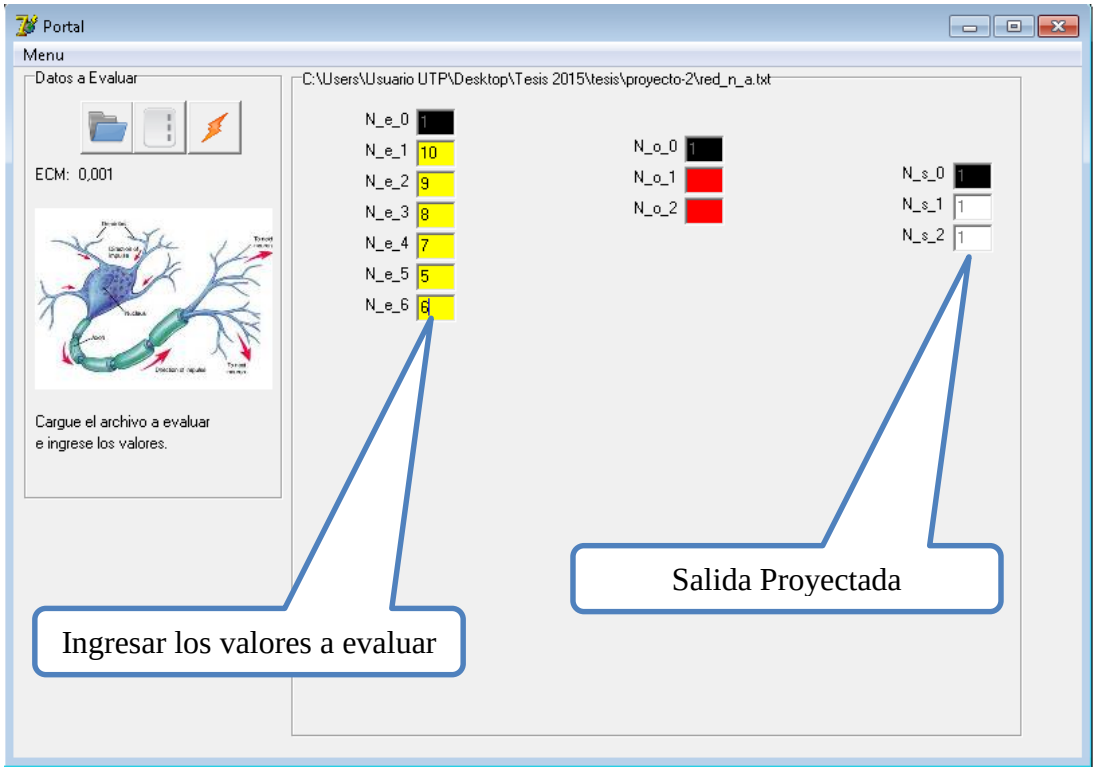
 *Limpiar todo y borrar la configuración hecha.*

 *Guardar red entenada*

Luego de entrenar la red puede descargar los pesos en un archivo y luego evaluarla.

 *Avanzar y registrar todos los datos de entrenamiento.*

4. *Módulo de Evaluación o Predicción:*



 *Cargar el archivo con los daos de la red*

 *Borrar datos cargados*

 *Evaluar la red*